

ORACLE®

ORACLE®

# Oracle Database 12c その他新機能

日本オラクル株式会社

ORACLE®  
DATABASE 12<sup>c</sup>



Plug into the **Cloud.**

以下の事項は、弊社の一般的な製品の方向性に関する概要を説明するものです。また、情報提供を唯一の目的とするものであり、いかなる契約にも組み込むことはできません。以下の事項は、マテリアルやコード、機能を提供することをコミットメント(確約)するものではないため、購買決定を行う際の判断材料になさらないで下さい。オラクル製品に関して記載されている機能の開発、リリースおよび時期については、弊社の裁量により決定されます。

OracleとJavaは、Oracle Corporation 及びその子会社、関連会社の米国及びその他の国における登録商標です。文中の社名、商品名等は各社の商標または登録商標である場合があります。

# Program Agenda

- Direct NFS(dNFS)関連の新機能
- デフォルト値関連の新機能
- Oracle SecureFiles (LOB) 関連の新機能
- その他新機能
  - データベース・スマート・フラッシュ・キャッシュ (DBSFC) での複数デバイスのサポート
  - SQL\*Loader Expressモード
  - OSのプロセッサ・グループとの統合
  - 非常に大きなネットワークバッファ
  - パターンマッチング

# Direct NFS(dNFS)関連の新機能



Plug into the **Cloud.**

# dNFS関連の新機能

1. dNFSクライアントの非同期I/O制御
2. SQL\*LoaderでのdNFSの使用
3. dNFSクライアントでのNFSバージョンの指定

# Direct NFS (dNFS) の概要

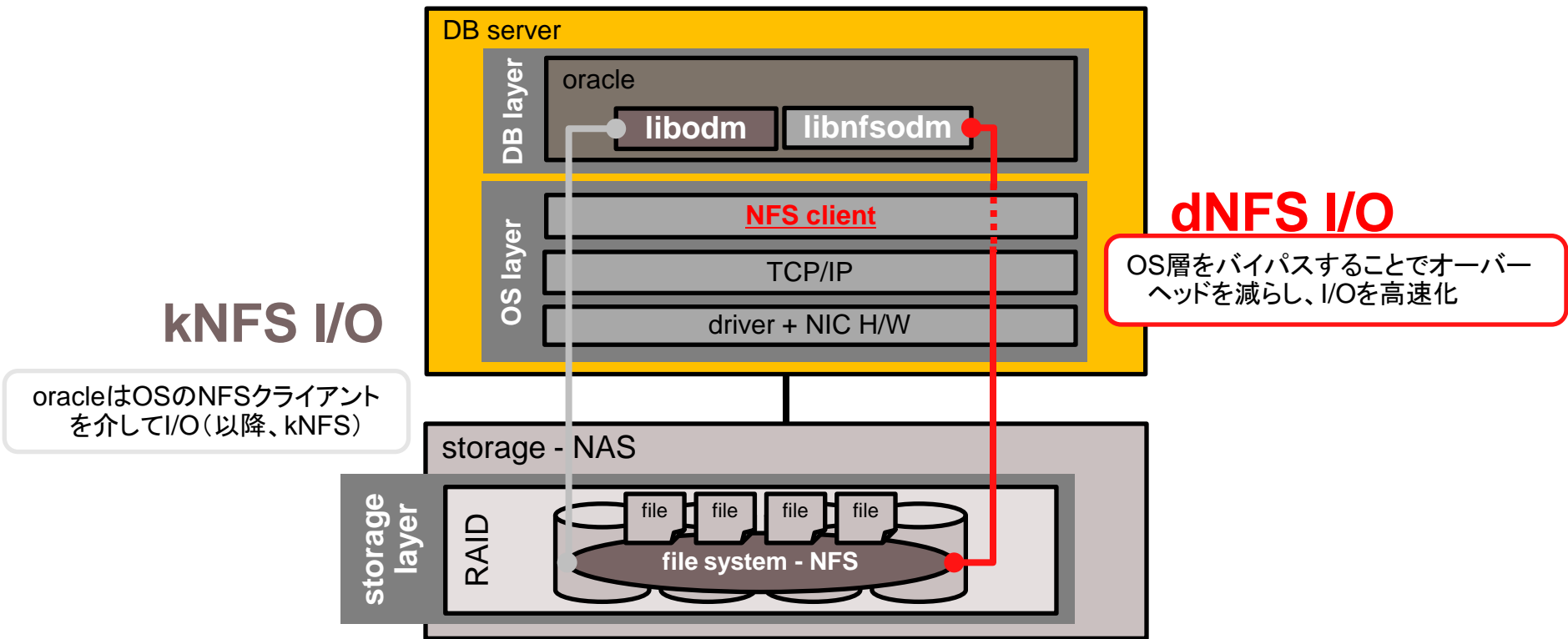
- dNFSとは

- Oracle Database内部に実装されたNFSクライアント機能
- Oracle Database 11g Release 1から使用可能

- dNFSの特徴

1. OSカーネルのNFSクライアント (kNFS) より高いディスクI/O性能
2. 簡単な手順で機能を有効化
  - アプリケーションの書き換えは必要ない
  - ストレージの構成や運用に影響はない
  - 複数イーサネット・ポートを使用したネットワーク帯域のスケーラビリティの設定が簡単

# Direct NFS (dNFS)の概要





# 構成に関する変更点

## RAC環境でのdNFSのデフォルト有効化

- Oracle Database 12cより、RAC環境ではdNFSの設定がデフォルトで有効化されている
  - Oracle Database 11gまでは、\$ORACLE\_HOME/rdbms/lib/ins\_rdbms.mkのmakeが必要だった
- RAC環境では、必要な設定はorafstabの作成のみとなる
- 無効化する場合は、Oracle DatabaseとGrid Infrastructureのそれぞれのホームで以下のコマンドを実行する

```
$ make -f ins_rdbms.mk dnfs_on
```

- ただし、シングルインスタンス環境やOracle restart環境では、デフォルトでは有効化されていないので、上記のmakeを実行する必要がある

```
$ make -f ins_rdbms.mk dnfs_off
```

# 1. dNFSクライアントの非同期I/O制御

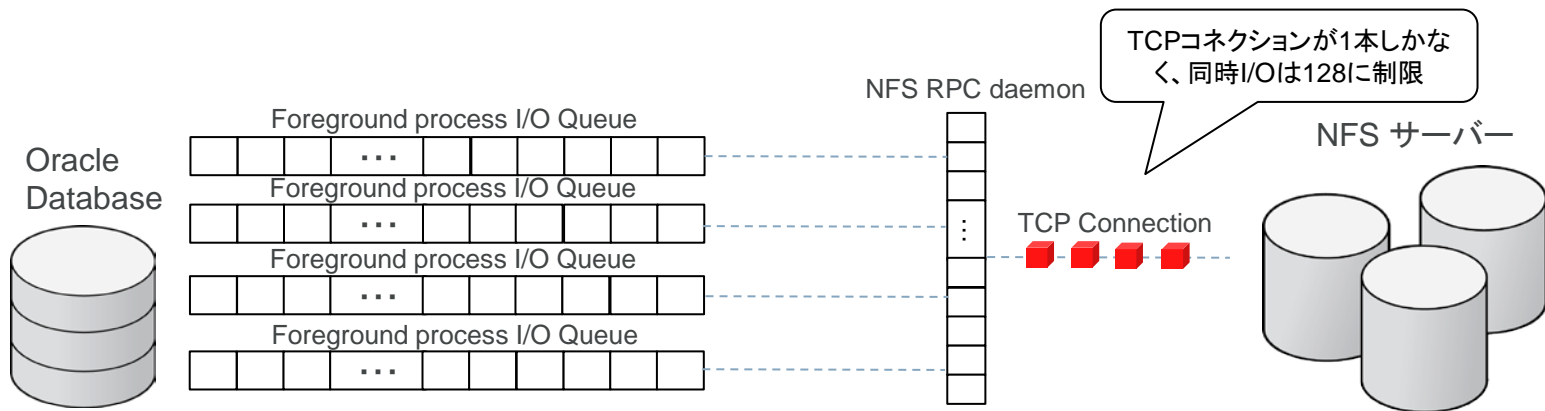
## 機能概要と設定方法

- I/Oが非常に多いワークロードが実行された場合、特定のNFSサーバーでオーバーロードが発生してしまう場合があった
- dNFSクライアント(Oracle Database)の発行するI/Oリクエストの数を制限し、NFSサーバーのパフォーマンスに合わせた最適な設定が可能に
- 設定方法
  - 初期化パラメータDNFS\_BATCH\_SIZEを変更する(デフォルト値:4096)
  - 値は0から4096の範囲で設定可能
    - alter system 文及びalter session文による動的な変更は不可
    - 変更を適用するためには、DBの再起動が必要

# 1. dNFSクライアントの非同期I/O制御

## kNFSのアーキテクチャ

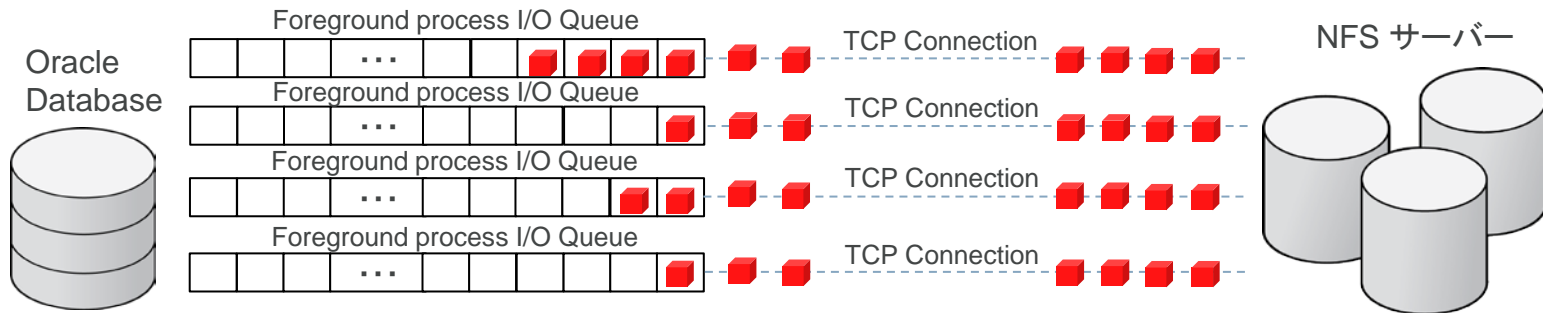
- kNFSでは、Oracle プロセスからのNFSサーバーへの接続は、OS上のRPC デーモンが実施する
  - OracleプロセスのRPCデーモンに対する同時I/Oは100、RPCデーモンからNFSサーバーへの同時I/Oは128に制限



# 1. dNFSクライアントの非同期I/O制御

## dNFSのアーキテクチャ

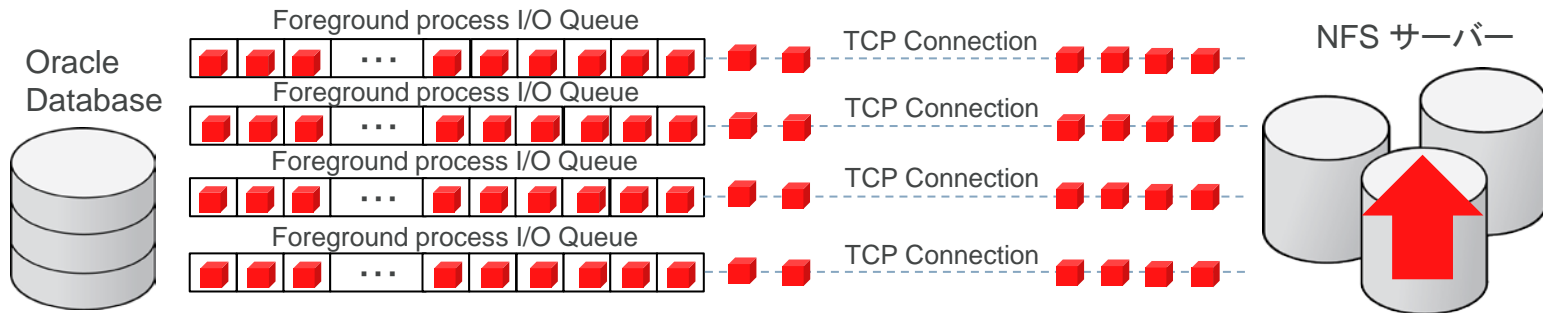
- dNFSでは、RPCデーモンをバイパスし、各dNFSクライアント(Oracle プロセス)がTCPコネクションをNFSサーバーに対して確立する
  - dNFSでは、1000までの並列処理が可能
  - RPCデーモンをバイパスすることで、foregroundプロセスが直接NFSサーバーとのやりとりを実施



# 1. dNFSクライアントの非同期I/O制御

## dNFSでの課題

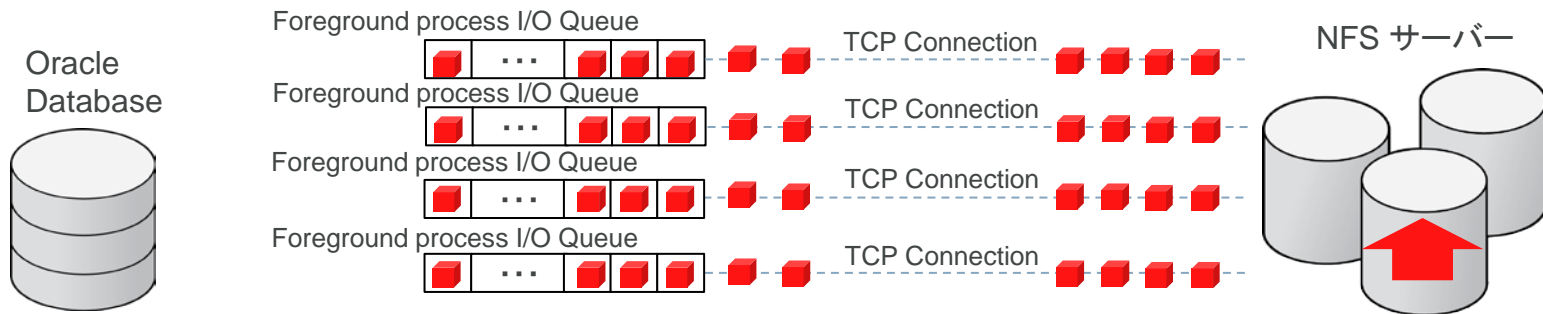
- foregroundプロセスが直接TCPコネクションを確立し、短時間に大量の非同期I/Oをすることで、NFSサーバーに対しての負荷が、kNFSのアーキテクチャと比較すると増えてしまう
  - コネクション数が増え、NFSサーバーに対するI/Oが増えれば、NFSサーバーの負荷も比例して増える



# 1. dNFSクライアントの非同期I/O制御

## dNFS側でのI/Oリクエスト数の制限

- foregroundプロセスが持つI/Oキューの数を制限し、foregroundプロセスからの非同期I/Oの同時大量リクエストの発生を低減させ、NFSサーバーのオーバーロードを防ぐ



# 1. dNFSクライアントの非同期I/O制御

## DNFS\_BATCH\_SIZEの設定指針

- 基本は、デフォルト値である4096で稼働させる
- この設定を変更する時は、NFSサーバー側にオーバーロードといった問題が発生した場合のチューニングオプション
  - 問題が発生した場合には、128に設定し、その後NFSサーバーのパフォーマンス状況に応じて、値の増減をさせる
  - 参考: Oracle Database Performance Tuning Guide 12c Release1  
18 Managing Operating System Resources

[http://docs.oracle.com/cd/E16655\\_01/server.121/e15857/pfgrf\\_os.htm#i4134](http://docs.oracle.com/cd/E16655_01/server.121/e15857/pfgrf_os.htm#i4134)

## 2. SQL\*Loader及び外部表でのdNFS

### 機能概要

- SQL\*Loader及び外部表の入力ファイルに対して、dNFSを使用したアクセスが可能
  - ファイルサイズに応じて自動でdNFSが利用される
    - 1GB以上の入力ファイルに対しては、自動でdNFSを使用したアクセスを実行
    - 1GB以下の入力ファイルに対しては、kNFSを使用したアクセスを実行
  - 明示的にdNFSによるアクセスを実行したい場合は、DNFS\_ENABLEパラメータをENABLEに指定する
- dNFSによる読み取りバッファ数に対して制限をかけることも可能
  - DNFS\_READBUFFERSパラメータの値を変更する



# SQL\*Loaderでの使用方法

## 明示的な設定方法

1. 明示的にdNFSを使用して入力ファイルにアクセスする場合

```
$ sqlldr username/password control=control_file DNFS_ENABLE=TRUE
```

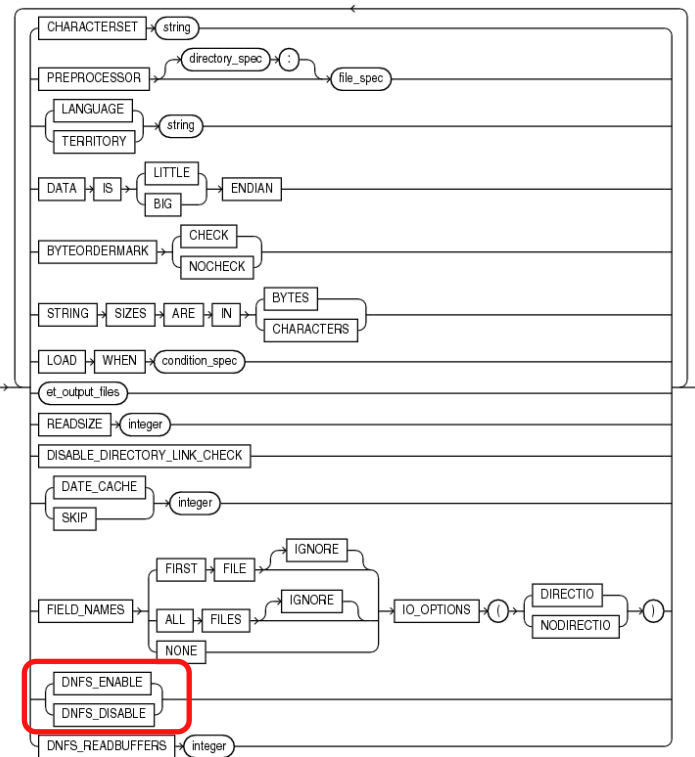
2. 明示的にdNFSを使用せずに入力ファイルにアクセスする場合

```
$ sqlldr username/password control=control_file DNFS_ENABLE=FALSE
```

# 外部表での使用方法

## 明示的な設定方法

- access パラメータのDNFS\_ENABLE/  
DNFS\_DISABLEで指定する
- 外部表でdNFSが利用されるのは、  
ORACLE\_LOADERを使用した場合
  - ORACLE\_DATAPUMPでは使用できない



### 3. dNFSクライアントでのNFSバージョンの指定

- oranfstab内で使用するNFSのバージョンを指定することが可能
  - Oracle Database 11g R2までは、NFS v3のみ
  - Oracle Database 12c より、NFS v3に加えて、NFS v4/NFS v4.1をサポート
    - NFSサーバーでこれらのプロトコルを有効になっている必要がある
  - 設定方法
    - oranfstab内のnfs\_versionパラメータで指定する

oranfstabの設定例

```
server : nfserver1
local: 192.168.0.20
path: 192.168.0.22
nfs_version: nfsv4
dontroute
export:/vol/dnfs mount:/mnt/dbfs
```

# デフォルト値関連の新機能

ORACLE<sup>®</sup> 12<sup>c</sup>  
DATABASE



Plug into the **Cloud.**

ORACLE<sup>®</sup>

# デフォルト値関連の新機能

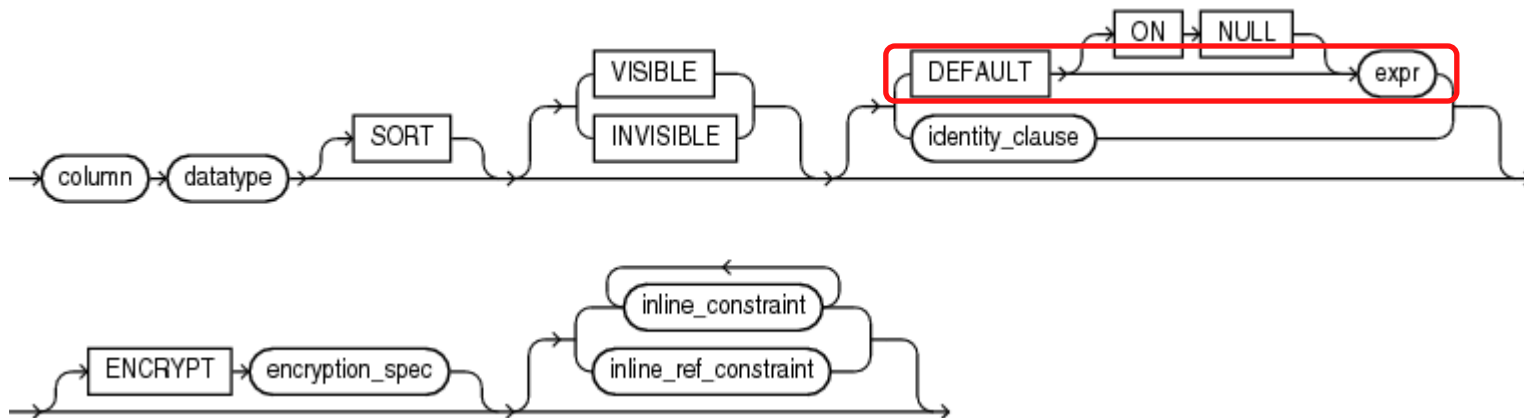
1. シーケンスに基づく列のデフォルト値
2. 列のデフォルト定義でのNULLのデフォルト値
3. メタデータのみでのデフォルト値
4. Identity列の導入

# 1. シーケンスに基づく列のデフォルト値

- Oracle Database 12cより、列のデフォルト値として、シーケンスを指定可能
  - デフォルト値として指定することで、実際にinsertされた時に採番が行われる
    - 従来までは、insert文の中でしか採番をすることができなかった
- 設定方法
  - create tableやalter table文の列定義のdefault句でシーケンス名を指定する
    - 事前にシーケンスを作成しておく必要がある
  - CURRVALとNEXTVALを指定可能
  - シーケンスの属性などは従来までの設定を使用可能

# 1. シーケンスに基づく列のデフォルト値

- 構文 1/2



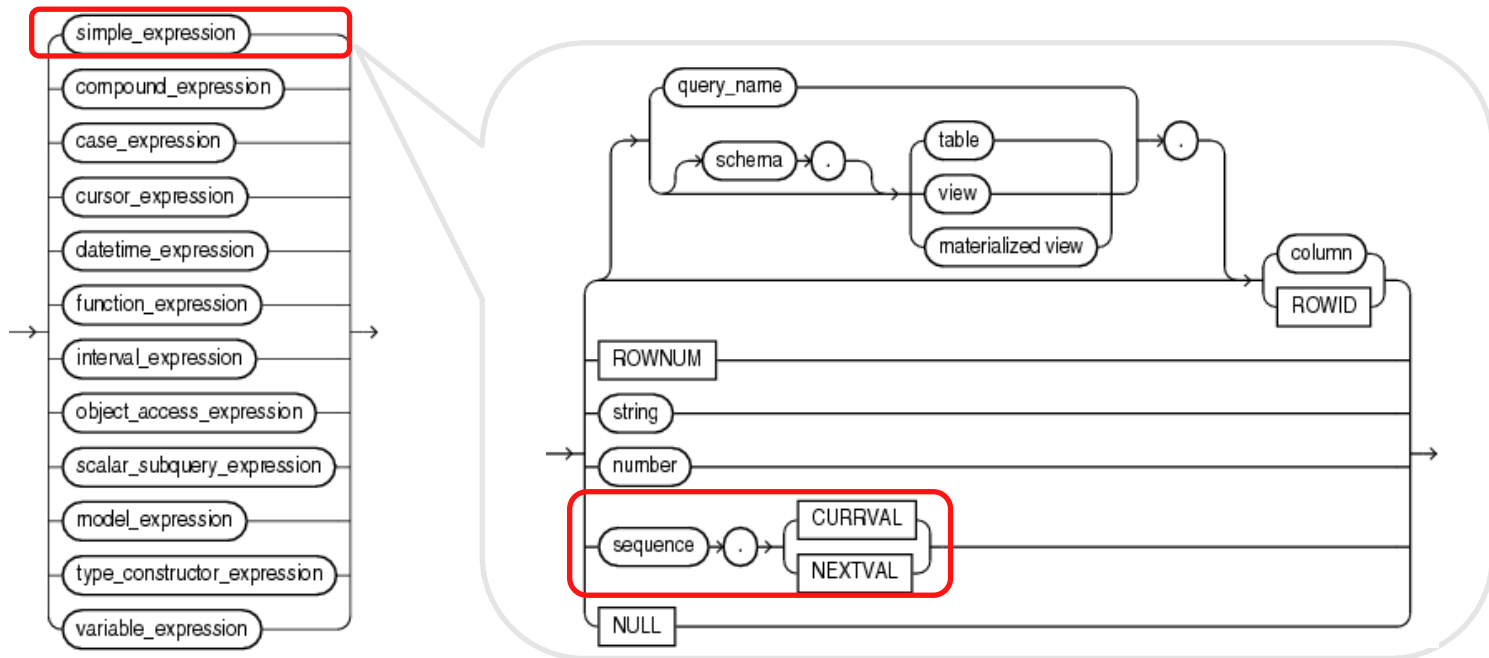
– 参考: Oracle Database SQL Language Reference 12c Release 1

[http://docs.oracle.com/cd/E16655\\_01/server.121/e17209/statements\\_7002.htm#i2095331](http://docs.oracle.com/cd/E16655_01/server.121/e17209/statements_7002.htm#i2095331)

[http://docs.oracle.com/cd/E16655\\_01/server.121/e17209/statements\\_3001.htm#CJAHHIBI](http://docs.oracle.com/cd/E16655_01/server.121/e17209/statements_3001.htm#CJAHHIBI)

# 1. シーケンスに基づく列のデフォルト値

## ■ 構文2/2





# 1. シーケンスに基づく列のデフォルト値 作成例

```
SQL> create sequence test_seq;
```

```
SQL> create table seq_test_tab  
(col1 int default test_seq.nextval primary key,  
 col2 varchar2(30))  
/
```

デフォルト値として先に作成したシーケンスを指定可能

```
SQL> insert into seq_test_tab(col2) values ('test');  
SQL> commit;
```

col1はnullでinsertを実行

```
SQL> select * from seq_test_tab;  
COL1 COL2
```

```
-----  
1 test
```

デフォルト値として、シーケンスから採番される

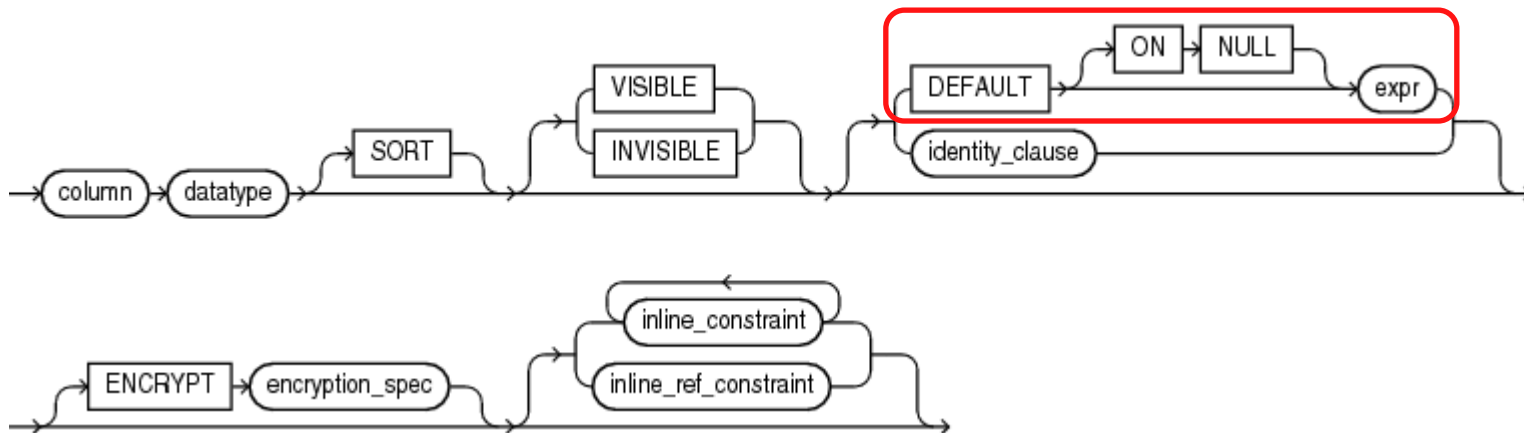
## 2. 列のデフォルト定義でのNULLのデフォルト値

- NULLを明示的にinsertする時に、デフォルト値として指定された値に置き換える機能
  - ON NULL句を列定義で指定する
  - ON NULL句を指定すると、NOT NULL制約とNOT DEFERABLE制約が自動的に指定される
    - NOT DEFERABLE制約: 制約に違反しているかどうかをトランザクションではなく、SQL文単位でチェックを行う

## 2. 列のデフォルト定義でのNULLのデフォルト値

### 構文

- デフォルト定義でNULLのデフォルト値の構成は以下の構文



- 参考: Oracle Database SQL Language Reference 12c Release 1

[http://docs.oracle.com/cd/E16655\\_01/server.121/e17209/statements\\_7002.htm#i2095331](http://docs.oracle.com/cd/E16655_01/server.121/e17209/statements_7002.htm#i2095331)

[http://docs.oracle.com/cd/E16655\\_01/server.121/e17209/statements\\_3001.htm#CJAHHIBI](http://docs.oracle.com/cd/E16655_01/server.121/e17209/statements_3001.htm#CJAHHIBI)

# 2. 列のデフォルト定義でのNULLのデフォルト値

## on null句による動作の違い

### on null句を設定しない場合の結果

```
SQL> create table test_tab  
(col1 number,  
col2 varchar2(10) default 'AAAAAA');
```

Table created.

```
SQL> insert into test_tab(col1) values (1);
```

1 row created.

```
SQL> commit;
```

Commit complete.

```
SQL> select * from test_tab;
```

```
COL1 COL2  
-----  
1 AAAAAA
```

```
SQL> insert into test_tab values (2,null);
```

1 row created.

```
SQL> commit;
```

Commit complete.

```
SQL> select * from test_tab;
```

```
COL1 COL2  
-----  
1 AAAAAA  
2
```

### on null句を設定した場合の結果

```
SQL> create table test_tab_on_null  
(col1 number,  
col2 varchar2(10) default on null 'AAAAAA');
```

表が作成されました。

```
SQL> insert into test_tab_on_null(col1) values (1);
```

1行が作成されました。

```
SQL> commit;
```

コミットが完了しました。

```
SQL> select * from test_tab_on_null;
```

```
COL1 COL2  
-----  
1 AAAAAA
```

```
SQL> insert into test_tab_on_null values (2,null);
```

1行が作成されました。

```
SQL> commit;
```

コミットが完了しました。

```
SQL> select * from test_tab_on_null;
```

```
COL1 COL2  
-----  
1 AAAAAA  
2 AAAAAA
```

デフォルト句のみの場合だと、nullが明示的にinsertされた時にはnullとなるが、on null句を使用する場合、nullがinsertされてもデフォルト値に置き換わる

# 3. メタデータのみでのデフォルト値

## 機能概要

- Oracle Database 11g R1より、デフォルト値を設定した列を追加した時に、メタデータにのみデフォルト値を格納し、データブロックには変更を加えないことで処理を高速化
  - 実際にデータファイルに書き込みを行わないため、領域の使用量も低下する
  - Oracle Database 11g R2までは、追加する列にデフォルト値に加え、**NOT NULL制約が必要だった**
- Oracle Database 12cでは、**NOT NULL制約は必要なく**、デフォルト値を持つ列の追加時には、メタデータのみの変更
  - デフォルト値以外のデータをinsertした場合やデフォルト値以外のデータでupdateした場合に、実際に領域が獲得される

# 4. Identity Column

## 機能概要

- ANSI準拠のIDENTITY column を実装
- 11gまでは、数値型の一意のIDをカラムとして定義する場合、事前にシーケンスを作成し、カラムのデフォルト値としてその順序を指定
- Identity Column機能により、事前のシーケンス作成は不要

```
SQL> create table t1
  2  (c1 number GENERATED BY DEFAULT ON NULL AS IDENTITY,
  3  c2 varchar2(10));
```

Table created.

```
SQL> insert into t1(c2) values('abc');
```

1 row created.

```
SQL> insert into t1(c1, c2) values(null, 'xyz');
```

1 row created.

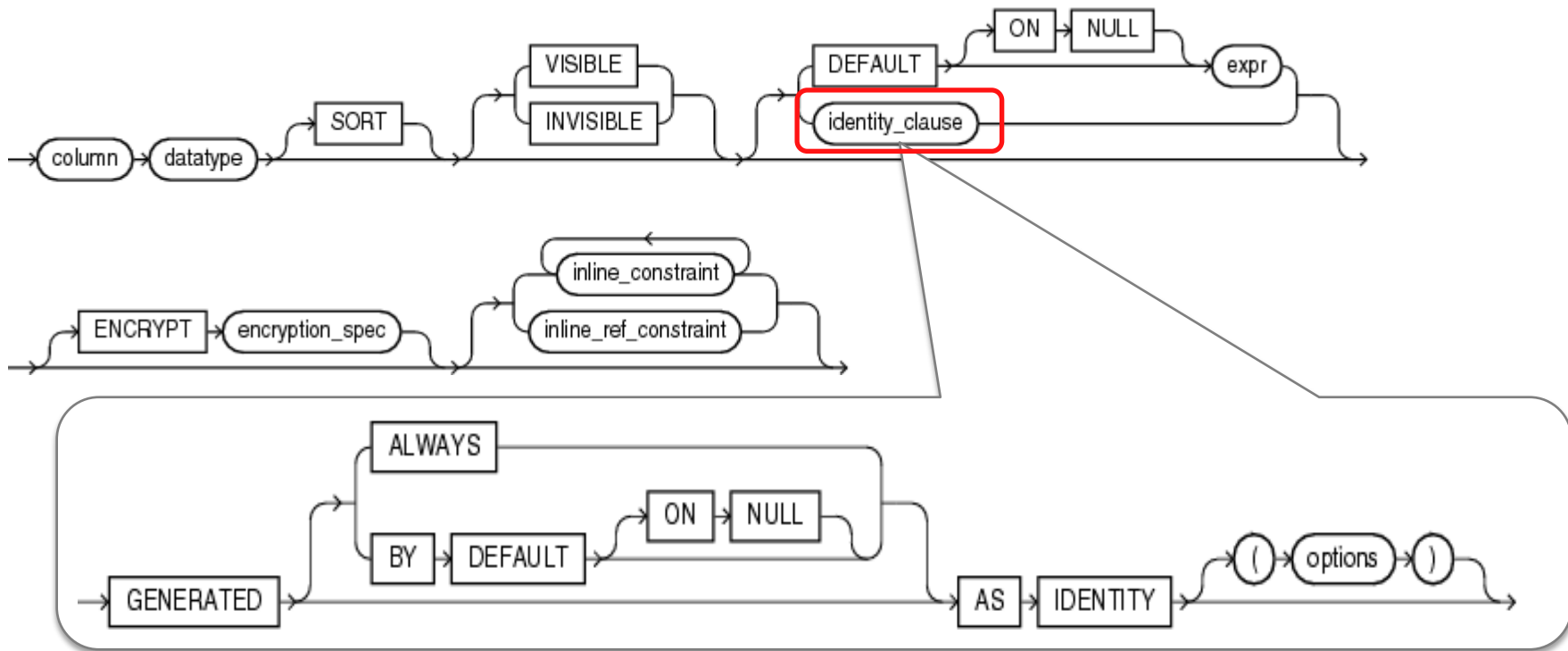
```
SQL> select c1, c2 from t1;
```

C1	C2
1	abc
2	xyz



# 4. Identity Column

## 構文 (1/2)

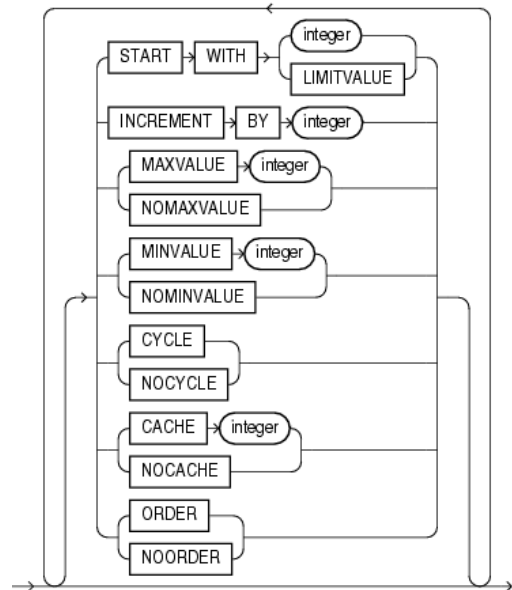


# 4. Identity Column

## 構文 (2/2)

options::=

```
[START WITH (<sequence generator start value> | LIMIT VALUE)
 | INCREMENT BY <sequence generator increment>
 | ( MAXVALUE <sequence generator max value> | NO MAXVALUE )
 | ( MINVALUE <sequence generator min value> | NO MINVALUE )
 | ( CYCLE | NO CYCLE )
 | ( CACHE integer | NOCACHE )
 | ( ORDER | NOORDER)]+
```



– 参考: Oracle Database SQL Language Reference 12c Release 1

[http://docs.oracle.com/cd/E16655\\_01/server.121/e17209/statements\\_7002.htm#i2095331](http://docs.oracle.com/cd/E16655_01/server.121/e17209/statements_7002.htm#i2095331)

[http://docs.oracle.com/cd/E16655\\_01/server.121/e17209/statements\\_3001.htm#CJAHHIBI](http://docs.oracle.com/cd/E16655_01/server.121/e17209/statements_3001.htm#CJAHHIBI)



# 5. Identity Column 設定例(1)

## ALWAYS の例

```
SQL> create table id_test(  
2 id number GENERATED ALWAYS AS IDENTITY,  
3 ename varchar2(30)  
4 );
```

```
SQL> select * from id_test;
```

ID	ENAME
1	user2

```
SQL> insert into id_test(ID, ENAME) values(100, 'user1');  
insert into id_test(ID, ENAME) values(100, 'user1')
```

\*

行1でエラーが発生しました。:

ORA-32795: GENERATED ALWAYSで作成されたアイデンティティ列には挿入できません

```
SQL> insert into id_test(ENAME) values('user2');
```

1行が作成されました。

# 5. Identity Column 設定例(2)

## BY DEFAULT の例

```
SQL> create table id_test2(  
2 ID number GENERATED BY DEFAULT AS IDENTITY,  
3 ENAME varchar2(30)  
4 );
```

```
SQL> select * from id_test2;
```

ID	ENAME
100	user1
1	user2

```
SQL> insert into id_test2(ID, ENAME) values(100, 'user1');
```

1行が作成されました。

```
SQL> insert into id_test2(ENAME) values('user2');
```

1行が作成されました。

# 5. Identity Column

## 制限事項

- 表に対して1つのみ定義可能
- 設定できるカラムのデータ型は数値型
- Identity Columnを設定したカラムにはDEFAULT句は指定不可
- Identity Columnを設定したカラムには暗黙的に NOT NULL 制約と NOT DEFERRABLE 制約が付加されるため、競合する制約は定義不可
  
- Identity Columnを暗号化する場合、暗号化アルゴリズムが推測されやすくなるため、強力な暗号化アルゴリズムの設定を推奨
- CTAS では Identity Column は継承されない

# Oracle SecureFiles (LOB) 関連の新機能

ORACLE<sup>®</sup> 12<sup>c</sup>  
DATABASE



Plug into the **Cloud.**

ORACLE<sup>®</sup>

# Oracle SecureFiles の新機能

1. Oracle SecureFilesのデフォルト化
2. Oracle DataPump でのLOB記憶域の変更
3. Oracle SecureFiles でのパラレルDMLの拡張
4. データ型の最大長の拡張

# 1. Oracle SecureFilesのデフォルト化

- Oracle Database 12cより、LOBのデフォルトがSecureFilesに変更
  - 初期化パラメータdb\_securefileがPREFERREDがデフォルト設定に変更 (COMPATIBLE パラメーターが12.0.0.0以上に設定されている場合)
  - storage句内でBASICFILEが明示的に指定されている/表領域が手動セグメント領域管理の場合は、BASICFILEとして作成される

## 2. Oracle DataPumpでのLOB記憶域の変更

- DataPump(impdp)で、インポートするLOBをSecureFilesに変更可能
  - Oracle Database 11gまで:従来のリリースまでは、LOBをインポートする時にexpdpをしたLOBの属性のまましかインポートできなかった
  - Oracle Database 12cでは:TRANSFORMパラメータのLOB STORAGEオプションでLOBの属性を変更可能
- 設定方法
  - TRANSFORM=LOB\_STORAGE:[SECUREFILE | BASICFILE | DEFAULT | NO\_CHANGE]をimpdp時に指定する
  - デフォルトはNO\_CHANGE(変更しない)

# 2. Oracle DataPumpでのLOB記憶域の変更

## 実行例

```
SQL> select table_name,column_name,securefile from user_lobs;
```

TABLE_NAME	COLUMN_NAME	SEC
T1	C1	NO

```
$ expdp test/test directory=datapump_dir tables=t1  
...中略...
```

ジョブ"TEST"."SYS\_EXPORT\_TABLE\_01"が月 9月 2 17:59:43 2013 elapsed 0 00:00:24で正常に完了しました

```
$ impdp test/test directory=datapump_dir transform=LOB_STORAGE:SECUREFILE  
...中略...
```

ジョブ"TEST"."SYS\_IMPORT\_FULL\_01"が月 9月 2 17:33:35 2013 elapsed 0 00:00:02で正常に完了しました

```
SQL> select table_name,column_name,securefile from user_lobs;
```

TABLE_NAME	COLUMN_NAME	SEC
T1	C1	YES



### 3. Oracle SecureFiles でのパラレルDMLの拡張

- Oracle Database 11gでは、SecureFilesが格納されているパーティション表に対するパラレルDMLはサポート済み
- Oracle Database 12cでは、SecureFilesが格納されている非パーティション表に対しても、パラレルDMLを実行可能
  - 実行可能な操作の例
    - INSERT / INSERT AS SELECT / CREATE TABLE AS SELECT / DELETE / UPDATEなど

# 3. Oracle SecureFiles でのパラレルDMLの拡張

## 非パーティション表に対してinsert as select を実行した場合

```
create table doc_tab_archive (doc_id number, doc_content BLOB) lob(doc_content) store as securefile parallel;  
alter session enable parallel dml;  
insert /*+ append */ into doc_tab_archive select * from doc_tab_daily;
```

### Oracle Database 11g R2

PLAN\_TABLE\_OUTPUT

Plan hash value: 4044616095

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	INSERT STATEMENT		100	196K	3 (0)	00:00:01
1	LOAD AS SELECT	DOC_TAB_ARCHIVE				
2	TABLE ACCESS FULL	DOC_TAB_DAILY	100	196K	3 (0)	00:00:01

### Oracle Database 12c R1

PLAN\_TABLE\_OUTPUT

Plan hash value: 2309883253

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	TQ	IN-OUT	PQ Distrib
0	INSERT STATEMENT		100	196K	3 (0)	00:00:01			
1	PX COORDINATOR								
2	PX SEND QC (RANDOM)	:TQ10001	100	196K	3 (0)	00:00:01	Q1,01	P->S	QC (RAND)
3	LOAD AS SELECT	DOC_TAB_ARCHIVE					Q1,01	PCWP	
4	OPTIMIZER STATISTICS GATHERING		100	196K	3 (0)	00:00:01	Q1,01	PCWP	
5	PX RECEIVE		100	196K	3 (0)	00:00:01	Q1,01	PCWP	
6	PX SEND ROUND-ROBIN	:TQ10000	100	196K	3 (0)	00:00:01	Q1,00	S->P	RND-ROBIN
7	PX SELECTOR						Q1,00	SCWC	
8	TABLE ACCESS FULL	DOC_TAB_DAILY	100	196K	3 (0)	00:00:01	Q1,00	SCWP	

# 4. Extended Data Types

## 機能概要

- varchar2、nvarchar2、raw型において、最大長が32,767バイトまで拡張
- 利用するには下記の2つの初期化パラメータを設定する

```
compatible = '12.0.0.0.0'  
max_string_size = 'EXTENDED'
```

- ただし、EXTENDEDからSTANDARD(従来通りの最大データ長)の変更不可
  - 4000bytes以下のデータをinsert/updateした場合は、インラインLOBとして格納される
  - それ以上のサイズのデータの場合は、アウトラインLOBとして格納される
- デフォルトのLOBがSecureFilesの場合には、SecureFilesとして格納される

# 4. Extended Data Types

## 設定方法

- 初期化パラメータMAX\_STRING\_SIZEの変更方法 (Non-CDB/シングルインスタンスの場合)
  1. データベースをシャットダウンする
  2. アップグレードモードで起動する
  3. MAX\_STRING\_SIZEをEXTENDEDに変更する
  4. \$ORACLE\_HOME/rdbms/admin/utl32k.sqlを実行する
  5. データベースをノーマルモードで再起動する
- 他の構成については、リファレンスガイド記載の変更方法に従って変更してください。  
[http://docs.oracle.com/cd/E16655\\_01/server.121/e17615/refrn10321.htm#sthref419](http://docs.oracle.com/cd/E16655_01/server.121/e17615/refrn10321.htm#sthref419)

# 4. Extended Data Types

## 注意事項

- MAX\_STRING\_SIZEを変更することに伴い、いくつかのデータベース・オブジェクトに変更が加わることで、無効化されるオブジェクトが出る
  - 仮想列はvarchar2(4000)、nvarchar2(4000)またはraw(2000)型になる
  - 上記の変更に伴い、索引キーの長さ制限を超えた場合、ファンクション索引は使用できなくなる
    - 索引再構築時にORA-1450が発生する
  - ビューにVARCHAR2(4000)、NVARCHAR2(4000)またはRAW(2000)型の式列が含まれている場合、これらのビューは無効化される
  - マテリアライズド・ビューは新しいメタデータで更新され、VARCHAR2(4000)、4000バイトのNVARCHAR2またはRAW(2000)型の式列になる

# その他の新機能



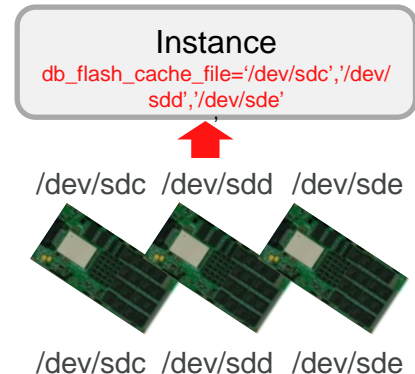
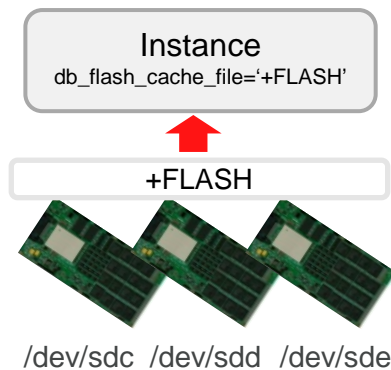
Plug into the **Cloud.**

# その他の新機能

1. データベース・スマート・フラッシュ・キャッシュ(DBSFC)での複数デバイスのサポート
2. SQL\*Loader Expressモード
3. プロセッサ・グループとの統合
4. 非常に大きなネットワークバッファ
5. パターンマッチング

# 1. データベース・スマート・フラッシュ・キャッシュの複数デバイスのサポート

- Oracle Database 11g R2では、複数のフラッシュデバイスをデータベース・スマート・フラッシュ・キャッシュとして使用する時には、直接複数デバイスを指定することはできなかった
  - 複数デバイスを利用する時は、フラッシュデバイスで構成されるASMディスク・グループを作成する/RAIDコントローラーで単一LUNとして構成する必要があった
- Oracle Database 12cから、データベース・スマート・フラッシュ・キャッシュに複数デバイスを指定できるように変更
  - 最大で16デバイスを指定可能
  - 従来通りのASMディスク・グループを作成し、指定する方法も可能





# 1. データベース・スマート・フラッシュ・キャッシュの複数デバイスのサポート

- 初期化パラメータdb\_flash\_cache\_fileに複数デバイスを指定する場合は、デバイス毎にDBSFCとして使用するファイルサイズを指定する
  - 設定例 (/dev/sdbを10GB、/dev/sdcを10GBを使用する場合)

```
orcl.db_flash_cache_file='/dev/sdb','/dev/sdc'  
orcl.db_flash_cache_size=10G,10G
```

- 以下のような設定の場合には、インスタンス起動時にalert.logに以下のようにエラーが出力される

```
orcl.db_flash_cache_file='/dev/sdb','/dev/sdc'  
orcl.db_flash_cache_size=10G
```

```
Different number of files and sizes in db_flash_cache_file 2 and db_flash_cache_size 1
```

# 2. SQL\*Loader Expressモード

## 概要

### SQL\*Loader

~ 12c

- 様々なデータをロード可能
  - 様々な形式の表・データに対応
  - パラメータによる多種多様なカスタマイズ



### SQL\*Loader Express

12c New

- シンプルなデータを簡単にロード
  - 制御ファイルの作成が不要
  - パラメータを指定したカスタマイズも可能
  - パラレル化による高速なロード

#### Negative

- 制御ファイル作成
- 手順が複雑
- パフォーマンスが出ない



# 2. SQL\*Loader Expressモード

## 使用方法

- コマンド

- ユーザ名・表名の指定のみでのデータのロード(制御ファイルは不要)

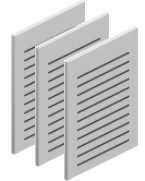
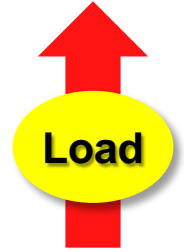
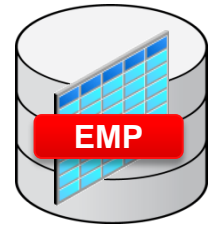
```
$ sqlldr scott/tiger table=EMP
```

- 実行時に必要なファイル

- ロードするデータのみ

- 各列のデータは「,」で区切り、各行は「改行」で変更(デフォルトの場合)
- ファイル名は「<表名>.dat」で作成(デフォルトの場合)

```
7369,SMITH,CLERK,7902,1980-12-17 00:00:00,800,,20  
7499,ALLEN,SALESMAN,7698,1981-02-20 00:00:00,1600,300,30  
7521,WARD,SALESMAN,7698,1981-02-22 00:00:00,1250,500,30  
.....
```

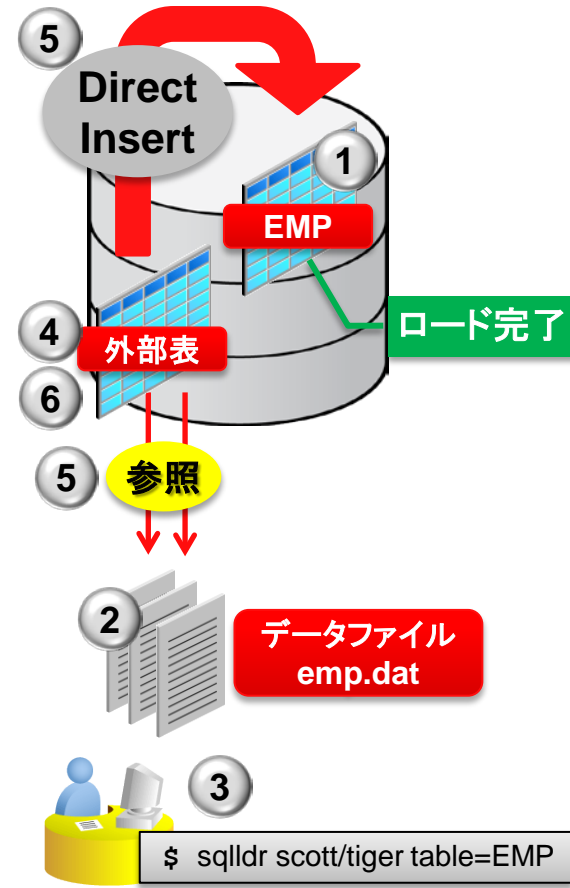


emp.dat

## 2. SQL\*Loader Expressモード

### デフォルト動作

- 外部表を経由したロード
  - ロード対象のテーブルを作成し、読み込みファイルを配置し、SQL\*Loader Expressモードを起動(1、2及び3)
  - 内部的に外部表を作成(4)
  - 作成した外部表を使用して、ロードするデータをダイレクト・インサートによりロード(5)
  - ロード完了後、4で作成した外部表を削除(6)
- ロードを実行せず、SQL文だけの生成も可能
  - EXTERNAL\_TABLE パラメータで制御
    - EXECUTE (デフォルト) : 外部表を使用してロード
    - NOT\_USED : 従来型モードでロード(従来のSQL\*Loaderでのロード)
    - GENERATE\_ONLY : SQL文のみ生成



# 2. SQL\*Loader Expressモード

## デフォルト動作

- データファイルの指定
  - カレントディレクトリの「<表名>.dat」を読み込む
  - DATA パラメータで任意のファイルの読み込みも可能
    - 単一ファイルの読み込み

```
$ sqlldr scott/tiger table=EMP data=emp.dat
```

- 複数ファイルの読み込み

- 複数ファイルのデータを一括ロード

```
$ sqlldr scott/tiger table=EMP data='emp*.dat'
```

```
$ sqlldr scott/tiger table=EMP data='emp?.dat'
```

```
$ sqlldr scott/tiger table=EMP data='emp1.dat','emp2.dat'
```

```
$ sqlldr scott/tiger table=EMP data=emp1.dat data=emp2.dat
```

任意の値・文字列を含むファイル  
(ex) emp.dat, emp1.dat, emp10.dat

任意の値(1文字)を含むファイル  
(ex) emp1.dat, emp2.dat, emp3.dat

# 2. SQL\*Loader Expressモード

## 出力ファイル

- 実行時に以下のファイルが出力される
  - <表名>.log
    - SQL\*Loader 実行時のログファイル
    - SQL\*Loader から実行されている SQL 文も参照可能
  - <表名>\_<実行プロセスのPID>.log\_xt
    - Database 上での処理実行時のログファイル
    - エラー発生時にはそのエラー情報を出力
  - <表名>\_<PID>.bad
    - ロードできなかったデータ
    - ロード時にエラーが発生した場合のみ出力される
    - DATA パラメータ指定時は「<最初のDATAファイル名>\_<PID>.bad」となる



```
$ sqlldr scott/tiger table=EMP
```

- emp.log
- emp\_21300.log\_xt
- emp\_21300.bad

```
$ sqlldr scott/tiger table=EMP  
DATA=emp1.dat, emp2.dat
```

- emp.log
- emp\_21301.log\_xt
- emp\_21302.log\_xt
- emp\_21301.bad
- emp\_21302.bad

## 2. SQL\*Loader Expressモード

### パラメータ

- DEGREE\_OF\_PARALLELISM
  - パラレル度を設定、デフォルト AUTO でパラレル化
- DIRECT
  - ダイレクト・ロードの使用可否を設定、デフォルト外部表ロードのため FALSE
- BAD
  - bad ファイルの出力ディレクトリ・名前を指定
- SILENT
  - 画面上への出力メッセージを制御
- DATA
  - インプットファイルの指定

## 2. SQL\*Loader Expressモード

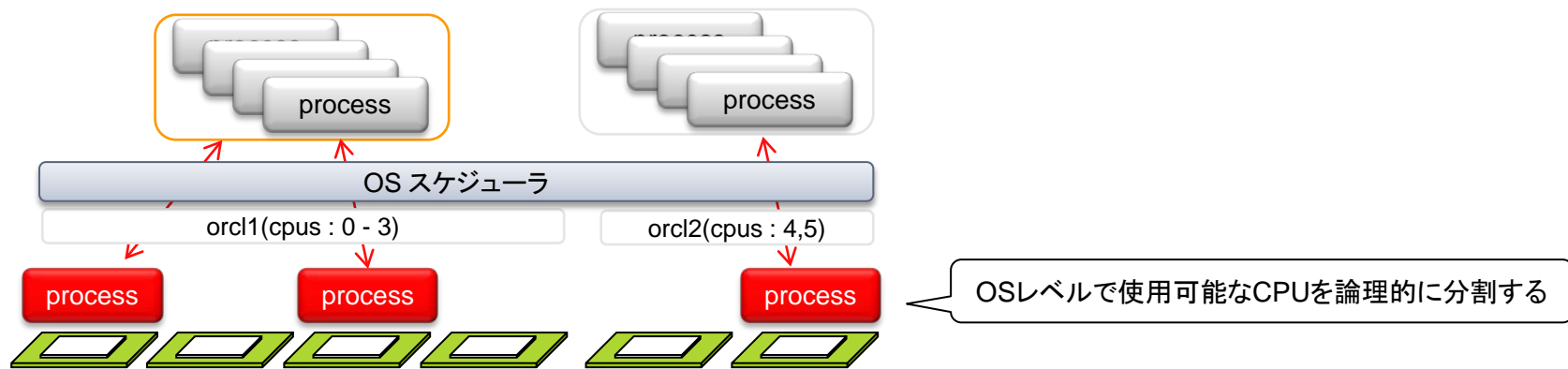
### パラメーター一覧

- **BAD**
- CHARACTERSET
- CSV
- **DATA**
- DATE\_FORMAT
- **DEGREE\_OF\_PARALLELISM**
- **DIRECT**
- DNFS\_ENABLE
- DNFS\_READBUFFERS
- ENCLOSED\_BY
- EXTERNAL\_TABLE
- FIELD\_NAMES
- NULLIF
- OPTIMALLY\_ENCLOSED\_BY
- PARFILE
- **SILENT**
- **TABLE** 必須
- TERMINATED\_BY
- TIMESTAMP\_FORMAT
- TRIM
- **USERID** 必須



### 3. プロセッサ・グループとの統合

- 初期化パラメータPROCESSOR\_GROUP\_NAMEでインスタンスが稼働するプロセッサ・グループ名を指定することが可能
  - インスタンスのプロセスが指定されたプロセッサ・グループ内のCPU上で稼働する
  - Linuxでは、cgroup(2.6.32カーネル以上)、Solarisでは、リソース・プール(Solaris 11 SRU4)に対応



# 3. プロセッサ・グループとの統合

## 設定方法と確認方法

- 設定方法

1. \$ORACLE\_HOME/rdbms/install/setup\_processor\_group.shを使用してプロセッサ・グループを作成する
2. 初期化パラメータPROCESSOR\_GROUP\_NAMEに1.で指定したグループ名を指定し、再起動する

- 確認方法

- alter.logに以下のようなメッセージが出力される

```
Sat Aug 10 15:27:13 2013
```

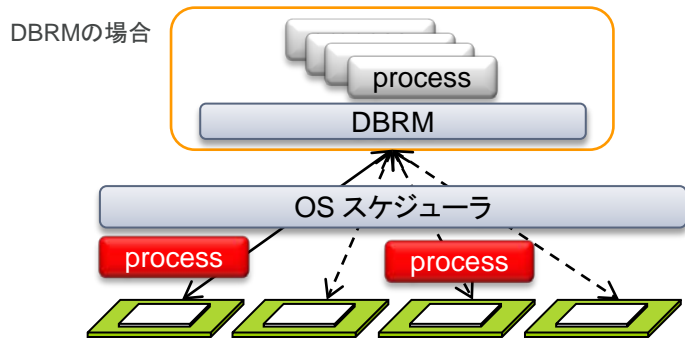
```
Instance has been started in processor group ORCL (NUMA Nodes: 0 CPUs: 0)
```

- ※ ASMインスタンスでの利用は不可

# 3. プロセッサ・グループとの統合

## データベース・リソース・マネージャーとの違い

- データベース・リソース・マネージャー(DBRM)は、Oracle Database内でのCPU使用率の調整
  - どのコアで稼働するかはOS側の制御にゆだねられる
- プロセッサ・グループはOS(カーネル)でのリソース制御
  - カーネルでグループに対するCPU割り当てを制御する



DBRMでは、Oracle Database内で、(CPU\_COUNT/OSのCPU数)で計算される値までしかCPUを使用しない  
実際にどのCPUが使用されるのまでは定義できない



cgroupやresource pool では、OS(カーネル)のレイヤーで、CPUリソースをグループ毎に分割する  
実際にどのCPUを使用するかというところまで定義可能



# ***DEMONSTRATION***

# デモンストレーション

## デモンストレーションの流れ

1. PROCESSOR\_GROUP\_NAMEを設定せずにswingbenchを実行
2. プロセッサ・グループの作成
3. PROCESSOR\_GROUP\_NAMEに作成したプロセッサ・グループを指定し、再起動
4. 再度swingbenchの実行し、使用されるCPUの違いを確認する

# 1. PROCESSOR\_GROUP\_NAMEを設定せずにswingbenchを実行

- Swingbenchを実行し、topでCPUの使用率を確認する

```
top - 16:49:29 up 13 days, 2:07, 4 users, load average: 0.51, 2.20, 5.68
Tasks: 486 total, 2 running, 484 sleeping, 0 stopped, 0 zombie
Cpu0 : 30.1%us, 7.8%sy, 0.0%ni, 58.3%id, 3.9%wa, 0.0%hi, 0.0%si, 0.0%st
Cpu1 : 13.7%us, 4.9%sy, 0.0%ni, 76.5%id, 4.9%wa, 0.0%hi, 0.0%si, 0.0%st
Cpu2 : 42.3%us, 9.3%sy, 0.0%ni, 48.5%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Cpu3 : 6.1%us, 3.1%sy, 0.0%ni, 89.8%id, 1.0%wa, 0.0%hi, 0.0%si, 0.0%st
Cpu4 : 4.1%us, 5.1%sy, 0.0%ni, 90.8%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Cpu5 : 4.0%us, 5.0%sy, 0.0%ni, 91.0%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Cpu6 : 2.0%us, 2.9%sy, 0.0%ni, 94.1%id, 0.0%wa, 0.0%hi, 0.0%si, 1.0%st
Cpu7 : 1.9%us, 3.7%sy, 0.0%ni, 87.9%id, 6.5%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 6164556k total, 5436904k used, 727652k free, 43992k buffers
Swap: 6111228k total, 1075688k used, 5035540k free, 3330840k cached
```

The screenshot shows the SwingBench 2.4.0.845 interface. At the top, it displays 'SwingBench 2.4.0.845 : "Order Entry (PLSQL)"'. Below this, there are several panels:

- Jobs Table:** A table listing various benchmark jobs with columns for Id, Class Name, Short Name, Load Ratio, and Activate.
 

Id	Class Name	Short Name	Load Ra...	Activate...
Customer Registr...	com.dom.benchmarking.swingbench.plsqltrans...NCR	20	<input checked="" type="checkbox"/>	
Browse Products	com.dom.benchmarking.swingbench.plsqltrans...BP	60	<input checked="" type="checkbox"/>	
Order Products	com.dom.benchmarking.swingbench.plsqltrans...OP	50	<input checked="" type="checkbox"/>	
Process Orders	com.dom.benchmarking.swingbench.plsqltrans...PO	5	<input checked="" type="checkbox"/>	
Browse Orders	com.dom.benchmarking.swingbench.plsqltrans...BO	10	<input checked="" type="checkbox"/>	
Sales Rep Query	com.dom.benchmarking.swingbench.plsqltrans...SQ	5	<input checked="" type="checkbox"/>	
- 3D Bar Chart:** A 3D bar chart titled 'Transactions Per Minute' showing the performance of different jobs. The Y-axis represents Transactions Per Minute, and the X-axis lists jobs like 'Customer Registration' and 'Warehouse'. The chart shows that 'Customer Registration' has the highest volume of transactions per minute.
- Summary Statistics:** At the bottom right, it shows 'トランザクション数/min : 7657' (Transactions/min: 7657) and 'ログオンしたユーザー数 : 150' (Logged on users: 150).

## 2. プロセッサ・グループの作成

- prepareの実行

```
$ $ORACLE_HOME/rdbms/install/setup_processor_group.sh -prepare
```

- プロセッサ・グループの作成

```
$ $ORACLE_HOME/rdbms/install/setup_processor_group.sh -create -name ORCL -cpus 1  
Successfully added CPUs 1 to processor group ORCL
```

To start an Oracle instance in this processor group, add the following to the init.ora parameter file:

```
PROCESSOR_GROUP_NAME=ORCL
```

## 2. プロセッサ・グループの作成

- 作成されたプロセッサ・グループの確認

```
$ $ORACLE_HOME/rdbms/install/ setup_processor_group.sh -show  
This is a NUMA system.
```

```
NUMA configuration:
```

```
-----
```

```
Node 0
```

```
CPUs: 0-7
```

```
Memory Total: 6299200 kB Free: 3767528 kB
```

```
Processor groups on this system:
```

```
-----
```

```
ORCL
```

```
nodes: 0
```

```
cpus: 1
```



# 3.PROCESSOR\_GROUP\_NAMEに作成した プロセッサ・グループを指定し、再起動

```
$ sqlplus / as sysdba
```

```
SQL*Plus: Release 12.1.0.1.0 Production on 金 8月 23 17:15:21 2013
```

```
Copyright (c) 1982, 2013, Oracle. All rights reserved.
```

```
SQL> alter system set processor_group_name=orcl scope=spfile;
```

システムが変更されました。

```
SQL> shutdown immediate
```

データベースがクローズされました。

データベースがディスマウントされました。

ORACLEインスタンスがシャットダウンされました。

```
SQL> startup
```

ORA-32004: obsolete or deprecated parameter(s) specified for RDBMS instance

ORACLEインスタンスが起動しました。

Total System Global Area 1920831488 bytes

Fixed Size 2289736 bytes

Variable Size 654315448 bytes

Database Buffers 1258291200 bytes

Redo Buffers 5935104 bytes

データベースがマウントされました。

データベースがオープンされました。

```
SQL> show parameter processor_group_name
```

NAME	TYPE	VALUE
processor_group_name	string	ORCL

# 4.再度swingbenchの実行し、使用されるCPUの違いを確認する

```
top - 17:39:46 up 13 days, 2:58, 4 users, load average: 3.21, 4.42, 2.43
Tasks: 567 total, 3 running, 564 sleeping, 0 stopped, 0 zombie
Cpu0 : 0.0%us, 0.0%sy, 0.0%ni, 99.0%id, 1.0%wa, 0.0%hi, 0.0%si, 0.0%st
Cpu1 : 28.7%us, 11.7%sy, 0.0%ni, 45.7%id, 13.8%wa, 0.0%hi, 0.0%si, 0.0%st
Cpu2 : 0.0%us, 0.0%sy, 0.0%ni, 100.0%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Cpu3 : 0.0%us, 0.0%sy, 0.0%ni, 100.0%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Cpu4 : 0.0%us, 0.0%sy, 0.0%ni, 100.0%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Cpu5 : 1.0%us, 2.9%sy, 0.0%ni, 96.1%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Cpu6 : 0.0%us, 0.0%sy, 0.0%ni, 100.0%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Cpu7 : 1.0%us, 1.0%sy, 0.0%ni, 98.0%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 6164556k total, 6014808k used, 149748k free, 14756k buffers
Swap: 6111228k total, 1071388k used, 5039840k free, 3231768k cached
```

The screenshot shows the SwingBench 2.4.0.845 application window titled "Order Entry (PLSQL)". It features a table of transactions and a 3D bar chart.

Id	Class Name	Short N.	Load Ra.	Activate...
Customer Registr...	com.dom.benchmarking.swingbench.plsqltrans...NCR	20		<input checked="" type="checkbox"/>
Browse Products	com.dom.benchmarking.swingbench.plsqltrans...BP	60		<input checked="" type="checkbox"/>
Order Products	com.dom.benchmarking.swingbench.plsqltrans...OP	50		<input checked="" type="checkbox"/>
Process Orders	com.dom.benchmarking.swingbench.plsqltrans...PO	5		<input checked="" type="checkbox"/>
Browse Orders	com.dom.benchmarking.swingbench.plsqltrans...BO	10		<input checked="" type="checkbox"/>
Sales Rep Query	com.dom.benchmarking.swingbench.plsqltrans...SQ	5		<input type="checkbox"/>

The 3D bar chart, titled "Transactions Per Minute", shows the performance of various transactions. The Y-axis represents the number of transactions per minute, and the X-axis lists transaction types like "Customer Registrat..." and "Warehouse...". The chart shows that "Customer Registrat..." has the highest volume, followed by "Warehouse...".

At the bottom right of the chart area, the following statistics are displayed:

- トランザクション数/min : 7578
- ログインしたユーザ数 : 150

# 4. 非常に大きなネットワークバッファ

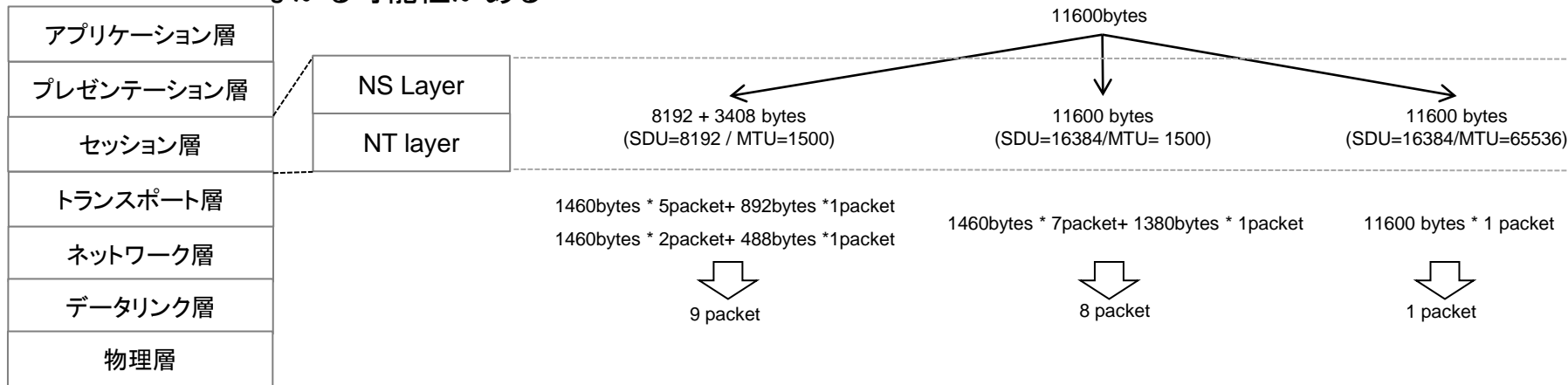
## SDUサイズの最大値の拡張

- SDUのサイズを最大で2097152bytes (2MB)に設定可能
  - デフォルトは8192bytes (8k)
  - Oracle Database 11g R2でのSDU最大値は65535bytes (64k)
- SDUの変更が効果的なケース
  - サーバーから戻されるデータが個別の packets に分かれる場合
  - 遅延の起こる広域ネットワーク(WAN)上にいる場合
  - パケット・サイズが一定している場合
  - 戻されるデータ量が多い場合

# 4. 非常に大きなネットワークバッファ

## ネットワーク通信とSDUの関連性

- SDUが大きくなることで、NS layerで分割されるNS Packetのサイズが変わる
  - NS layer : 上位プロトコルから受け取ったデータをSDUの設定に基づき、NS Packetに分割
  - NT layer : NS layerから受け取ったNS Packetを下位レイヤーに引き渡す
- SDUでの分割が少なくなれば、下位レイヤーでのパケット数が減り、ネットワーク通信の最適化につながる可能性がある

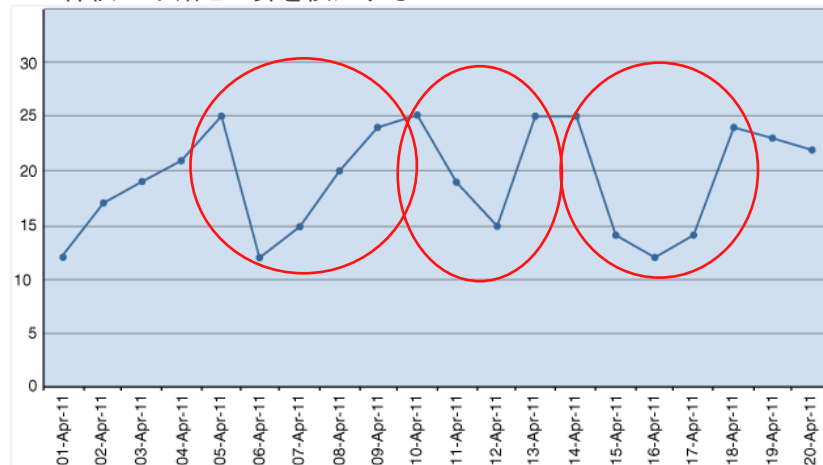


# 5. パターンマッチング

## 機能概要

- データの値ではなく、値の変化に対して、該当する行を検索する
  - とらえたいデータ変化のパターン(前日と比較して値がどう推移したかなど)を検索する
    - 株価の変動に伴う取引や、犯罪件数の変化の検知、クレジットカードの不正利用検知など
  - “*match\_recognize*”句を使用する
- 動作概要
  - 行を論理的なサブセットに分割し、順序付けを行う
  - 正規表現を利用したパターン変数を定義
  - 定義したパターン変数が各行に対してマッチするかを判断
    - 定義したそれぞれのパターンが個々の行や集計の結果の条件として使用される

株価の下落と上昇を検知する



# 5. パターンマッチング

## 構文

- PARTITION BY : 行を論理グループに分割する
- ORDER BY : 分割された論理グループ内で順序付けを行う
- MEASURES : 出力表の列を定義する
- PER MATCH : 一致ごとにサマリー表示するか詳細表示するかを定義する
- AFTER MATCH SKIP : 一致が見つかった後の再開場所を定義する
- PATTERN : 一致する行パターンを定義する
- DEFINE : PATTERN句で指定するパターンを定義する

```
SELECT *
FROM Ticker MATCH_RECOGNIZE (
  PARTITION BY symbol
  ORDER BY tstamp
  MEASURES STRT.tstamp AS start_tstamp,
            LAST(DOWN.tstamp) AS bottom_tstamp,
            LAST(UP.tstamp) AS end_tstamp,
            MATCH_NUMBER() AS match_num,
            CLASSIFIER() AS var_match

  ALL ROWS PER MATCH
  AFTER MATCH SKIP TO LAST UP
  PATTERN (STRT DOWN+ UP+)
  DEFINE
    DOWN AS DOWN.price < PREV(DOWN.price),
    UP AS UP.price > PREV(UP.price)
) MR
ORDER BY MR.symbol, MR.match_num, MR.tstamp;
```

# 5. パターンマッチング

## クエリーの意味

シンボル列の値で論理グループを構成し、その中で  
tstamp列で順序付けを行う

DOWNパターンに該当した行のうち、最後の行の日付をbottom\_tstampと  
して、UPパターンに該当した行のうち、最後の行の日付をend\_tstampとし  
て表示する

一致した行が複数にわたる場合、1行ごとに出力をする

現在の一致の最後の行の次の行でパターン一致を再開する

DOWNパターンとUPパターンが1回以上続くパターンを検索する

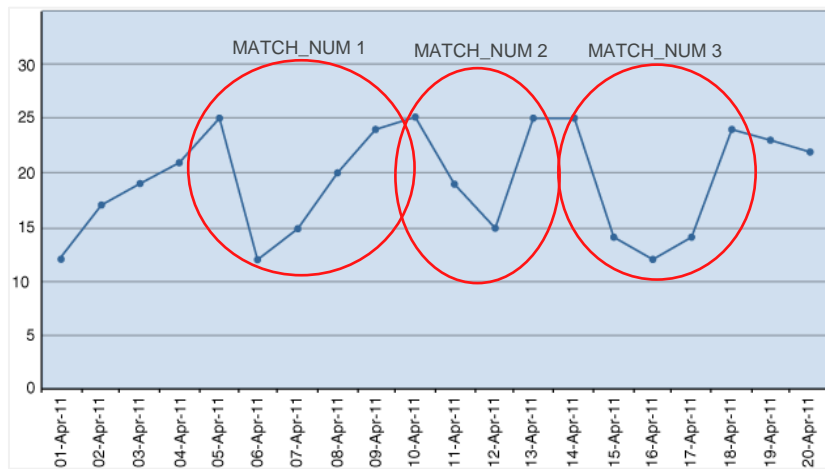
DOWNを前の行と比較してpriceがそれ以下の時、UPを  
前の行と比較して、priceがそれ以上の時と定義

```
SELECT *
FROM Ticker MATCH_RECOGNIZE (
  PARTITION BY symbol
  ORDER BY tstamp
  MEASURES STRT.tstamp AS start_tstamp,
            LAST(DOWN.tstamp) AS bottom_tstamp,
            LAST(UP.tstamp) AS end_tstamp,
            MATCH_NUMBER() AS match_num,
            CLASSIFIER() AS var_match
  ALL ROWS PER MATCH
  AFTER MATCH SKIP TO LAST UP
  PATTERN (STRT DOWN+ UP+)
  DEFINE
    DOWN AS DOWN.price < PREV(DOWN.price),
    UP AS UP.price > PREV(UP.price)
) MR
ORDER BY MR.symbol, MR.match_num, MR.tstamp;
```

# 5. パターンマッチング

## 実行例 その1

- 株価が前日と比較して下落した後、上昇したパターンを検出する



```
SELECT *
FROM Ticker MATCH_RECOGNIZE (
  PARTITION BY symbol
  ORDER BY tstamp
  MEASURES STRT.tstamp AS start_tstamp,
            LAST(DOWN.tstamp) AS bottom_tstamp,
            LAST(UP.tstamp) AS end_tstamp,
            MATCH_NUMBER() AS match_num,
            CLASSIFIER() AS var_match

  ALL ROWS PER MATCH
  AFTER MATCH SKIP TO LAST UP
  PATTERN (STRT DOWN+ UP+)
  DEFINE
    DOWN AS DOWN.price < PREV(DOWN.price),
    UP AS UP.price > PREV(UP.price)
) MR
ORDER BY MR.symbol, MR.match_num, MR.tstamp;
```

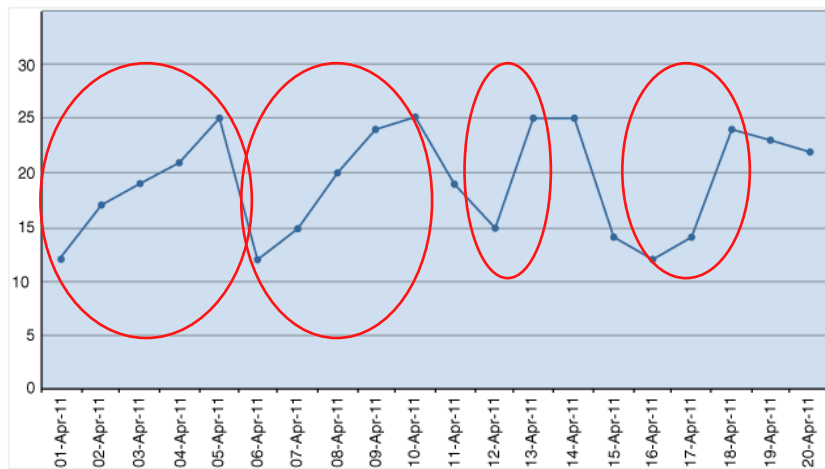
SYMBOL	TSTAMP	START_TS	BOTTOM_T	END_TSTA	MATCH_NUM	VAR_MATCH	PRICE
ORCL	05-08-11	05-08-11			1	STRT	25
ORCL	06-08-11	05-08-11	06-08-11		1	DOWN	12
ORCL	07-08-13	05-08-11	06-08-11	07-08-13	1	UP	15
ORCL	08-08-13	05-08-11	06-08-11	08-08-13	1	UP	20
ORCL	09-08-13	05-08-11	06-08-11	09-08-13	1	UP	24
ORCL	10-08-13	05-08-11	06-08-11	10-08-13	1	UP	25
ORCL	10-08-13	10-08-13			2	STRT	25
ORCL	11-08-13	10-08-13	11-08-13		2	DOWN	19
ORCL	12-08-13	10-08-13	12-08-13		2	DOWN	15
ORCL	13-08-13	10-08-13	12-08-13	13-08-13	2	UP	25
ORCL	14-08-13	14-08-13			3	STRT	25
ORCL	15-08-13	14-08-13	15-08-13		3	DOWN	14
ORCL	16-08-13	14-08-13	16-08-13		3	DOWN	12
ORCL	17-08-13	14-08-13	16-08-13	17-08-13	3	UP	14
ORCL	18-08-13	14-08-13	16-08-13	18-08-13	3	UP	24



# 5. パターンマッチング

## 実行例 その2

- 株価が前日の価格と比較して上昇したパターンを検出する



```
SELECT *
FROM Ticker MATCH_RECOGNIZE (
  PARTITION BY symbol
  ORDER BY tstamp
  MEASURES STRT.tstamp AS start_tstamp,
            LAST(UP.tstamp) AS up_tstamp,
            MATCH_NUMBER() AS match_num,
            CLASSIFIER() AS var_match
  ALL ROWS PER MATCH
  AFTER MATCH SKIP TO LAST UP
  PATTERN (STRT UP+)
  DEFINE
    UP AS UP.price > PREV(UP.price)
) MR
ORDER BY MR.symbol, MR.match_num, MR.tstamp;
```

SYMBOL	TSTAMP	START_TS	UP_TSTAM	MATCH_NUM	VAR_MATCH	PRICE
ORCL	01-08-11	01-08-11		1	STRT	12
ORCL	02-08-11	01-08-11	02-08-11	1	UP	17
ORCL	03-08-11	01-08-11	03-08-11	1	UP	19
ORCL	04-08-11	01-08-11	04-08-11	1	UP	21
ORCL	05-08-11	01-08-11	05-08-11	1	UP	25
ORCL	06-08-11	06-08-11		2	STRT	12
ORCL	07-08-13	06-08-11	07-08-13	2	UP	15
ORCL	08-08-13	06-08-11	08-08-13	2	UP	20
ORCL	09-08-13	06-08-11	09-08-13	2	UP	24
ORCL	10-08-13	06-08-11	10-08-13	2	UP	25
ORCL	12-08-13	12-08-13		3	STRT	15
ORCL	13-08-13	12-08-13	13-08-13	3	UP	25
ORCL	16-08-13	16-08-13		4	STRT	12
ORCL	17-08-13	16-08-13	17-08-13	4	UP	14
ORCL	18-08-13	16-08-13	18-08-13	4	UP	24

# 5. パターンマッチング

## パターンマッチングに関する詳細情報

- マニュアル
  - Oracle Database Data Warehousing Guide 12c Release 1 (12.1)  
18 SQL for Pattern Matching  
[http://docs.oracle.com/cd/E16655\\_01/server.121/e17749/pattern.htm#BABJBAA](http://docs.oracle.com/cd/E16655_01/server.121/e17749/pattern.htm#BABJBAA)

**Hardware and Software**

**ORACLE®**

**Engineered to Work Together**

ORACLE®