Administração de Banco de Dados

Aula 6

Prof. Marcos Alexandruk

Aula 6

Estruturas lógicas do Oracle:

Tabelas

- **tabelas relacionais**
- **tabelas temporárias**
- **tabelas organizadas por índices**
- **tabelas de objetos**
- **tabelas externas**
- **tabelas clusterizadas**
- **tabelas particionadas**

© 2017 - Prof. Marcos Alexandruk

Tabelas

- As tabelas são as unidades básicas de armazenamento do banco de dados Oracle. Tabelas são estruturas bidimensionais compostas por linhas (tuplas) e colunas (campos).
- O banco dados Oracle apresenta diferentes tipos de tabelas para atender a necessidades ou demandas específicas:
  - tabelas relacionais;
  - tabelas temporárias;
  - tabelas organizadas por índices;
  - tabelas de objetos;
  - tabelas externas;
  - tabelas clusterizadas;
  - tabelas particionadas.

© 2017 - Prof. Marcos Alexandruk

Tabelas

Tabelas relacionais

- As tabelas relacionais são as mais comuns em um banco de dados.
- Tabelas relacionais ou HOT (Heap Organized Table) apresentam linhas que não são organizadas em nenhuma ordem específica.
- É possível especificar no comando CREATE TABLE a cláusula ORGANIZATION HEAP, porém, como este é o padrão, torna-se desnecessário incluir esta cláusula.

© 2017 - Prof. Marcos Alexandruk

Tabelas

Tabelas relacionais

- **Exemplo:**

```
CREATE TABLE TABELA_1 (
COLUNA_1 NUMBER (4),
COLUNA_2 VARCHAR2 (20) )
TABLESPACE USERS;
```

© 2017 - Prof. Marcos Alexandruk

Tabelas

Tabelas temporárias

- As tabelas temporárias, disponíveis a partir do Oracle 8i, são temporárias somente com relação aos dados que armazenam, pois sua estrutura, isto é, as denominações das colunas e seus respectivos tipos de dados e tamanhos, ficam gravadas no banco de dados até que ocorra sua alteração através de um comando ALTER TABLE ou sua eliminação através de um comando DROP TABLE.
- Qualquer usuário com permissões suficientes pode acessar as tabelas temporárias, utilizar instruções SELECT ou comandos DML (INSERT, UPDATE e DELETE), no entanto, cada usuário poderá ver apenas os seus dados. Quando utilizar o comando TRUNCATE TABLE, apenas os seus dados serão removidos da tabela.
- Há também duas cláusulas que controlam o modo como os dados serão removidos da tabela temporária. A cláusula (padrão) ON COMMIT DELETE ROWS remove as linhas da tabela quando é emitido um comando COMMIT ou ROLLBACK e a cláusula ON COMMIT PRESERVE ROWS mantém as linhas na tabela mesmo após um comando COMMIT ou ROLLBACK, porém, ao final da sessão todos os dados do usuário serão removidos da tabela temporária.

© 2017 - Prof. Marcos Alexandruk

Tabelas temporárias

- **Exemplo 1: Dados removidos após o COMMIT:**

```
CREATE GLOBAL TEMPORARY TABLE TABELA_3A (
COLUNA_1 NUMBER (4),
COLUNA_2 VARCHAR2 (20) );

INSERT INTO TABELA_3A VALUES (1,'ABC');

SELECT * FROM TABELA_3A;

COMMIT;

SELECT * FROM TABELA_3A;
```

© 2017 - Prof. Marcos Alexandruk

Tabelas temporárias

- **Exemplo 2: Dados mantidos após o COMMIT e removidos no final da sessão:**

```
CREATE GLOBAL TEMPORARY TABLE TABELA_3B (
COLUNA_1 NUMBER (4),
COLUNA_2 VARCHAR2 (20) )
ON COMMIT PRESERVE ROWS;

INSERT INTO TABELA_3B VALUES (1,'ABC');

SELECT * FROM TABELA_3B;

COMMIT;

SELECT * FROM TABELA_3B;
```

© 2017 - Prof. Marcos Alexandruk

Tabelas

Tabelas organizadas por índice

- A criação de um índice diminui o tempo necessário para localização de uma linha específica na tabela. Uma tabela organizada por índice (IOT - Index Organized Table) armazena as linhas de uma tabela em um índice de árvore B (B-tree). Cada nó contém uma coluna indexada juntamente com uma ou mais colunas não indexadas. Uma tabela organizada por índice deve, portanto, ter sempre uma chave primária.

- Uma das principais vantagens deste tipo de tabela é que apenas uma estrutura de armazenamento precisa ser atualizada e não duas como seria o caso se fosse criada uma tabela relacional e separadamente um índice.

© 2017 - Prof. Marcos Alexandruk

Tabelas organizadas por índice

Para melhor compreensão vamos tomar como exemplo uma tabela na qual o campo CODIGO, Primary Key, do tipo NUMBER, apresenta valores ordenados de 1 a 500.

Admitamos que os valores entre 200 e 300 precisem ser alterados para outra faixa, entre 600 e 700.

Em uma tabela HOT (Heap Organized Table) a alteração ocorrerá apenas nos valores do campo, não haverá alteração do endereço da linha (ROWID).

Em uma IOT (Index Organized Table) a alteração será física, no nível de armazenamento dos dados, nos blocos de dados. A ROWID se movimentará para a organizar a Primary Key numa sequência lógica. Portanto, poderá haver melhora de performance em consultas utilizando o campo Primary Key (CODIGO, conforme exemplo).

Resumindo: uma OIT não necessita de uma tabela de índices, como em uma HOT, ela já se comporta movimentando as linhas para que a organização da mesma não necessite de uma estrutura a parte.

© 2017 - Prof. Marcos Alexandruk

Tabelas

Tabelas organizadas por índice

- **Exemplo:**

```
CREATE TABLE TABELA_3 (
COLUNA_1 NUMBER (4),
COLUNA_2 VARCHAR2 (20),
CONSTRAINT TABELA_3_PK PRIMARY KEY (COLUNA_1) )
ORGANIZATION INDEX;
```

© 2017 - Prof. Marcos Alexandruk

Tabelas

Tabelas HOT X Tabelas OIT

```
CREATE TABLE TABELA_3A (
COLUNA_1 NUMBER (4),
CONSTRAINT TABELA_3A_PK PRIMARY KEY (COLUNA_1) );

INSERT INTO TABELA_3A VALUES (1001),
INSERT INTO TABELA_3A VALUES (1002),
INSERT INTO TABELA_3A VALUES (1003),
COMMIT;

SELECT ROWID, COLUNA_1 FROM TABELA_3A;

UPDATE TABELA_3A SET COLUNA_1 = 1004 WHERE COLUNA_1 = 1001;

COMMIT;

SELECT ROWID, COLUNA_1 FROM TABELA_3A;
```

© 2017 - Prof. Marcos Alexandruk

Tabelas

Tabelas HOT X Tabelas OIT

```
CREATE TABLE TABELA_3A (
COLUNA_1 NUMBER (4),
CONSTRAINT TABELA_3A_PK PRIMARY KEY (COLUNA_1) )
ORGANIZATION INDEX;

INSERT INTO TABELA_3A VALUES (1001),
INSERT INTO TABELA_3A VALUES (1002),
INSERT INTO TABELA_3A VALUES (1003),
COMMIT;

SELECT ROWID, COLUNA_1 FROM TABELA_3A;

UPDATE TABELA_3A SET COLUNA_1 = 1004 WHERE COLUNA_1 = 1001;

COMMIT;

SELECT ROWID, COLUNA_1 FROM TABELA_3A;
```

© 2017 - Prof. Marcos Alexandruk

Tabelas de objetos

- As linhas das tabelas de objetos correspondem aos objetos propriamente ditos ou a instanciações de definições de tipos.
- As linhas em uma tabela de objetos podem ser referenciadas através da chave primaria, se esta existir, ou através do OID (Object Identifier) que é exclusivo para cada objeto.

© 2017 - Prof. Marcos Alexandruk

Tabelas

Tabelas de objetos

```
CREATE TYPE ALUNO_T AS OBJECT (
RA NUMBER (10),
NOME VARCHAR2 (30) );
/

CREATE TYPE TURMA_T AS OBJECT (
CODIGO NUMBER (4),
SALA NUMBER (3),
ALUNO ALUNO_T );
/

CREATE TABLE TURMA OF TURMA_T;

INSERT INTO TURMA VALUES (1001, 320,
ALUNO_T (111222333,'ANTONIO ALVARES') );
```

© 2017 - Prof. Marcos Alexandruk

Tabelas externas

- **As tabelas externas, disponíveis a partir do Oracle 9i, permitem que o usuário acesse uma fonte externa de dados, um arquivo de texto, por exemplo, como se fosse uma tabela. O conteúdo é armazenado externamente, apenas os metadados para a tabela são armazenados no dicionário de dados.**
- ***Não é possível criar índices nem executar operações de inserção, atualização ou exclusão de dados em tabelas externas.***

© 2017 - Prof. Marcos Alexandruk

Tabelas

Tabelas externas

Criar um arquivo CSV denominado cliente.csv, conforme segue:

```
1001,ANTONIO ALVARES
1002,BEATRIZ BARBOSA
```

Gravar o arquivo em uma pasta e criar um objeto DIRECTORY no Oracle:

```
CREATE OR REPLACE DIRECTORY teste AS 'C:\teste';
```

Conceder privilégio de leitura e escrita no objeto TESTE para um usuário do Oracle:

```
GRANT READ, WRITE ON DIRECTORY teste TO scott;
```

Criar uma tabela externa chamada CLIENTE:

```
CREATE TABLE CLIENTE (
ID NUMBER(4),
NOME varchar2(50) )
ORGANIZATION EXTERNAL (
TYPE ORACLE_LOADER
DEFAULT DIRECTORY TESTE
ACCESS PARAMETERS (
RECORDS DELIMITED BY NEWLINE
FIELDS TERMINATED BY ',' )
LOCATION ('cliente.csv') );
```

Ler os dados da tabela externa:

```
SELECT * FROM CLIENTE;
```

© 2017 - Prof. Marcos Alexandruk

Tabelas clusterizadas

- As tabelas clusterizadas podem melhorar o desempenho quando duas ou mais tabelas são frequentemente acessadas juntas. Reduzem também a quantidade de espaço necessário para armazenar as colunas que duas tabelas têm em comum.
- O valor da chave é armazenado em um índice de cluster (cluster index) que, assim como em um índice tradicional, aprimora as consultas às tabelas clusterizadas quando acessadas pelo valor da chave do cluster.

© 2017 - Prof. Marcos Alexandruk

Tabelas

Tabelas clusterizadas

```
CREATE CLUSTER EMP_DEPT_CLUSTER (
ID_DEPT NUMBER(4) )
SIZE 512;

CREATE INDEX IDX_EMP_DEPT_CLUSTER
ON CLUSTER EMP_DEPT_CLUSTER;

CREATE TABLE DEPARTAMENTO (
ID_DEPT NUMBER (4),
NOME_DEPT VARCHAR2 (20),
CONSTRAINT DEPARTAMENTO_PK PRIMARY KEY (ID_DEPT) )
CLUSTER EMP_DEPT_CLUSTER (ID_DEPT);

CREATE TABLE EMPREGADO (
ID_EMP NUMBER (4),
NOME_EMP VARCHAR2 (30),
ID_DEPT NUMBER (4),
CONSTRAINT EMPREGADO_PK PRIMARY KEY (ID_EMP),
CONSTRAINT EMP_DEPT_FK FOREIGN KEY (ID_DEPT) REFERENCES DEPARTAMENTO (ID_DEPT) )
CLUSTER EMP_DEPT_CLUSTER (ID_DEPT);
```

© 2017 - Prof. Marcos Alexandruk

Tabelas

Tabelas particionadas

- O particionamento permite decompor tabelas muito grandes e índices em partes menores melhorando assim o seu gerenciamento.
- Estas partes menores são denominadas partições. Cada partição é um objeto independente com seu próprio nome e, opcionalmente, suas próprias características de armazenamento.
- Portanto, se uma partição estiver em um volume de disco corrompido, as outras partições ainda estarão disponíveis para consultas enquanto o volume corrompido for reparado.
- O particionamento é transparente para a aplicação, isto é, não é necessário especificar explicitamente a partição quando se utiliza, por exemplo, a linguagem SQL para uma consulta ao banco de dados.

© 2017 - Prof. Marcos Alexandruk

Tabelas

Tabelas particionadas

```
CREATE TABLESPACE TBS_PART_ATE_1950
LOGGING DATAFILE 'C:\CLIENTE_1.DBF'
SIZE 1M
AUTOEXTEND ON NEXT 10M
MAXSIZE UNLIMITED
EXTENT MANAGEMENT LOCAL
SEGMENT SPACE MANAGEMENT AUTO;
```

```
CREATE TABLESPACE TBS_PART_1950_2000
LOGGING DATAFILE 'C:\CLIENTE_2.DBF'
SIZE 1M
AUTOEXTEND ON NEXT 10M
MAXSIZE UNLIMITED
EXTENT MANAGEMENT LOCAL
SEGMENT SPACE MANAGEMENT AUTO;
```

```
CREATE TABLESPACE TBS_PART_MAIOR_2000
LOGGING DATAFILE 'C:\CLIENTE_3.DBF'
SIZE 1M
AUTOEXTEND ON NEXT 10M
MAXSIZE UNLIMITED
EXTENT MANAGEMENT LOCAL
SEGMENT SPACE MANAGEMENT AUTO;
```

```
CREATE TABLE CLIENTE (
ID_CLIENTE NUMBER (4),
NOME_CLIENTE VARCHAR2 (30),
ANO_NASC NUMBER (4) NOT NULL,
CONSTRAINT CLIENTE_PK PRIMARY KEY (ID_CLIENTE) )
PARTITION BY RANGE (ANO_NASC)
(
PARTITION PART_ATE_1950 VALUES LESS THAN (1950) TABLESPACE TBS_PART_ATE_1950,
PARTITION PART_1950_2000 VALUES LESS THAN (2000) TABLESPACE TBS_PART_1950_2000,
PARTITION PART_MAIOR_2000 VALUES LESS THAN (MAXVALUE) TABLESPACE TBS_PART_MAIOR_2000
);
```

© 2017 - Prof. Marcos Alexandruk

Tabelas

- **Leituras recomendadas:**

https://docs.oracle.com/database/121/CNCPT/tablecls.htm#CNCPT010

http://www.fabioprado.net/2011/02/criando-tabelas-particionadas-para.html

http://www.oracle.com/technetwork/pt/articles/database-performance/criando-tabelas-particionadas-3047879-ptb.html

© 2017 - Prof. Marcos Alexandruk