**GENERALE**

**Concetti di base:**

Database + istanza si chiama RDBMS.

L’istanza contiene i processi e la memoria allocata

Il database sono i file su disco.

Un database può essere accessibile da più istanze(RAC)

Una istanza può accedere ad un solo database.

.

L’istanza è composta da almeno i seguenti processi:

**PMON**: monitora tutti gli altri processi e li riavvia in caso di errore, disconnette le sessioni in timeout, scarta le operazioni “utente” fallite

**SMON**: monitora il sistema e ottimizza aree di memoria fisica eliminando i segment non più utilizzati

**DBWN**: database writer, scrive i dati dalla RAM sull’HDD. Questo avviene ogni volta non c’è abbastanza spazio nella buffer cache per nuove operazioni o quando viene eseguito un checkpoint.

**LGWR**: Tiene aggiornati i redo Log scrivendo quello che c’è nel Redo-Buffer sul redo-log-file, questo avviene ogni 3 secondi,quando viene committata una transazione o prima che il DBWN scriva su disco

**CKPT**: checkpoint, tiene aggiornato il control file e dice al writer quando può scrivere. Quindi ogni volta che avviene un checkpoint i blocchi in ram vengono trasferiti su disco e viene aggiornato il control file con un numero di “versione” progressivo. Il checkpoint avviene ogni LOG\_CHECKPOINT\_TIMEOUT, oppure quando un redo-log raggiunge la capacità massima o manualmente con ALTER SYSTEM CHECKPOINT

**ARC:** Si occupa di trasferire i redo-log verso un archivio, inoltre in caso avessimo uno standby-DB, si occuperebbe anche di trasferire tutti i log delle transazioni a questo standby-db

**SGA Global System Area**

È un’area di memoria condivisa da tutti i processi oracle ed è composta da:

**Database buffer cache**: include i dati prelevati dal disco che devono essere letti o modificati. Numero di buffer chache: DB\_BLOCK\_BUFFERS, dimensione buffer chache: DB\_BLOCK\_SIZE

**Redo Log buffer:** contiene i dati dei redo log prima che vengano scritti su disco

**Shared pool:** contiene i dati già elaborati che potrebbero essere utili per successive elaborazioni. In pratica se faccio SQL statement, oltre che nella PGA viene messo nella shared pool in modo che le successive se sono simili saranno più veloci, questo comprende anche il risultato della query. Inoltre fa cache anche del dizionario.

**Large pool:** simile alla shared ma contiene dati troppo grandi per la shared. È utile quando c’è tanto I/O, oppure quando si fanno query molto complesse oppure per operazioni di Backup/Restore che richiederebbero molta memoria.

**Java Pool**: contiene i programmi java che vengono utilizzati nell’ambito DB

**Keep Buffer pool:** Se alcuni blocchi vengono letti molto spesso posso scegliere di “pinnarli” nella cache mettendoli nella keep buffer pool, in questo modo non verranno flushati quando viene richiesto spazio alla buffer cache.

**Recycle Buffer Pool:** è simile alla Keep Buffer pool ma qui si vanno a mettere tabelle molto grandi che però vengono utilizzate poco.

**PGA: Program Global Area**

È un area di memoria non condivisa, ogni processo di base dispone di una propria porzione di PGA.

Ogni query o DML che viene lanciato utilizza una PGA.

Flow di un comando lanciato dall’utente



**Database File:**

Il datafile è il file sul disco che contiene le tabelle, il datafile viene suddiviso in blocchi da 4/8k all’interno di ogni blocco ci sono le righe di una tabella. Ogni blocco ha anche sempre una porzione libera per gli aggiornamenti. Quindi se stiamo scrivendo su un blocco ma rimarrebbe poco spazio, oracle passa al blocco successivo.

Per migliorare le performance, Oracle per ogni tabella prenota un certo numero di blocchi contigui, questi blocchi contigui appartenenti alla stessa tabella si chiamano **extent**. Se la tabella satura l’attuale extent, oracle ne prenota un altro.

**Tablespace**: è un insieme di datafile, infatti quando creo una tabella Oracle crea un tablespace. Tablespace:LVM=datafile:disk

**SEGMENT**: in pratica sono tutti gli oggetti oracle che occupano spazio, tipo le tabelle

**CONTROL FILE**

È importantissimo infatti è multiplexed, ovvero scritto in più posti.

In particolare contiene le informazioni dei checkpoint, dei redo log e del log switching.

I file vengono scritti su disco ad ogni checkpoint, se il server si spegne brutalmente, al riavvio oracle fa partire un recovery che guarda l’ultimo checkpoint e con i redolog ricostruisce le operazioni successive.

I redo log sono più di uno, viene scritto il primo, quando si esaurisce passa all’altro(log switching) fino all’ultimo, dopo di che sovrascrive il primo.

Ad ogni log switching ci sarà un checkpoint, per non perdere le informazioni dalla sovrascrittura del log switching si può abilitare l’archive log.

**SPFILE**

Ci sono tutti i parametri di avvio di un DB, è un file binario quindi per modificarlo devo utilizzare l’SQL statement “ALTER SYSTEM”, mentre per vedere il contenuto “SHOW PARAMETER SPFILE”.

La sua posizione è in : $ORACLE\_HOME/dbs/spfile[SID].ora

Posso backupparlo con RMAN

17

**AVVIARE UNA ISTANZA**
Normalmente si utilizza SQLPlus, per lanciare l’utility: sqlplus / as sysdba

Fase di startup: La fase di startup si compone di questi step:

SHUTDOWN: istanza ferma e database non disponibile

NOMOUNT: istanza accesa e DB non disponibile

MOUNT: istanza ferma e DB disponibile solo per operazioni admin

OPEN: istanza attiva e DB disponibile per tutto.

Dallo stato iniziale di shutdown posso passare allo stato di open con il comando “startup”, se voglio invece passare a stati intermedi è “startup nomount” o “startup mount”

Se invece mi trovo già in uno stato di NOMOUNT, posso passare a mount o open con il comando “alter database open” oppure “alter database mount”

**SPEGNERE UNA ISTANZA**

Il comando per spegnere è shutdown, il quale eseguirà gli step inversi dello startup.

Ci sono vari tipi di shutdown:

shutdown(normale): non accetta nuove connessioni, attende che tutti gli utenti si disconnettono e poi chiude tutto.

Shutdown transactional:Non accetta nuove connessioni, Attende che gli utenti collegati salvino le modifiche, a quel punto li disconnette e spegne.

Shutdown immediate: esegue un checkpoint e butta fuori tutti, gli utenti potrebbero perdere le modifiche non salvate.

Shutdown abort: è come staccare la spina, sicuramente ci sarà un recovery alla riaccensione

**UTENTI E SCHEMI:**

Lo schema è un insieme di tabelle e altri oggetti, lo schema è di proprietà di un utente e ha lo stesso nome. Anche se concettualmente utenti e schemi sono diversi, per praticità possiamo immaginarli come la stessa cosa.

L’utente SYS è il più potente di tutti e andrebbe utilizzato raramente.

L’utente SYSTEM è meno potete ma comunque può amministrare quasi tutto, quindi viene normalmente utilizzato questo.

**Privilegi**

**SYSTEM PRIVILEGES:**

I privilegi possono essere di sistema come (create table, create tablespace, create database link, create synonim, create view, create session, unlimited tablespace)

Es: GRANT CREATE TABLE TO Pippo [with admin option] Se aggiungo “with admin option” quell’utente può dare lo stesso grant ad un altro utente.

Se do GRANT CREATE TABLE TO Pippo; Pippo può creare tabelle solo sul suo schema

Se do GRANT CREATE ANY TABLE TO Pippo; Pippo può creare tabelle su qualsiasi schema

**OBJECT PRIVILEGES:**

sono più specifici di un particolare oggetto, i più comuni sono: Select,update,delete,alter,execute

es: GRANT DELETE ON Pippo.Nometabella TO Pluto;

es: GRANT DELETE ON Pippo.Nometabella TO Pluto with grant option;

REVOKE INSERT ON pippo.tabella FROM pippo; Rimuove un object privilege a pippo e in cascata anche a chi ha dato il grant pippo.

REVOKE CREATE TABLE FROM pippo; Rimuove un SYSTEM Privilege a pippo, ma non in cascata a chi ha dato il GRANT pippo.

**RUOLI:**

I ruoli sono un set di GRANT, posso anche annidare un ruolo su un altro

CREATE ROLE manager\_ROLE; creo il ruolo

GRANT INSERT, UPDATE ON pippo.tabella TO manager\_ROLE; Assegno i grant

GRANT manager\_ROLE to pippo; Assegno il ruolo a pippo

GRANT manager\_ROLE to operation\_ROLE; Annido un ruolo

**PROFILE:**

I profili sono utilizzati per impostare parametri del kernel come CONCURRED\_SESSION, IDLE\_TIME ecc oppure policy password. Se non viene specificato un utente prende il profilo di default.

Select username, profile from dba\_users where username=’Pippo’; Vede il profilo di pippo

Select \* from dba\_profiles where profile=’DEFAULT’; vede i dettagli del profilo

CREATE PROFILE Pippo\_PROFILE LIMIT

 SESSION\_PER\_USER2

 IDLE\_TIME 5; Crea un profilo con queste impostazioni

Alter user Pippo profile pippo\_profile; assegna un profilo

**NETWORKING**

Sul Server girerà un listener, il listener di default è in ascolto sulla porta 1521, una volta che il client si collega al listener la comunicazione avverrà direttamente tra la PGA e il client.

Per avviare o stoppare il listener: lsnrctl start|stop

Il listener può lavorare in due modi:

Dedicated server: per ogni connessione il listener apre dei processi oracle dedicati(consuma più risorse)

Shared server: il listener passa la palla al processo dispatcher che si occuperà di assegnare delle risorse libere alla connessione in maniera dinamica(consuma meno risorse server)

Per configurare il listener tramite gui posso utilizzare NETCA

Il percorso del listener è $ORACLE\_HOME\network\admin\

**Connessione ad un DB**

Se ci si trova sul server stesso, basta lanciare sqlplus system/P@ssword

Se invece ci si trova su un client, si devono specificare 4 parametri:

IP/Nome del server

Nome del database

Protocollo utilizzato

Porta in ascolto sul server (di default 1521 TCP)

Senza utilizzare un tnsnames.ora, posso fare la connessione con questa stringa:

$ sqlplus pippo/pippo123@192.168.1.50:1521/DatabaseName

Quando il client si collega, la connessione viene presa in carico dal listener.

Sul client il programma utilizzato per connettersi, utilizzerà un file di testo chiamato tnsnames.ora, qui vengono definite delle stringhe di connessione ad esempio:

miastringadiconnessione = (DESCRIPTION =

(ADDRESS\_LIST =

(ADDRESS = (PROTOCOL = TCP)(HOST = servername)(PORT = 1521))

 )

(CONNECT\_DATA = (SERVICE\_NAME = MIODATABASE) ) )

Quindi per collegarsi si può utilizzare il comando:

sqlplus utente/P#assword@miastringadiconnessione

Posso creare un tnsname.ora anche da gui con il comando netmgr

Utilizzando un tnsname posso connettermi con:

sqlplus pippo/pippo123@NomeStringa

**Creare un utente:**

Create user corso identified by corsopwd; (crea l’utente e il relativo shema)

Grant connect, resource to corso; (assegna delle risorse di base per poter lavorare e fare tabelle)

**Comandi SqlPlus**

Show all: mostra tutti i parametri

Show parametro: mostra il valore del parametro

Set linesize 156: imposta il numero di linee orizzontali a 156

Desc nometabella: mostra una descrizione della tabella

Per eseguire un file .sql: start comando.sql oppure @comando.sql

Per cambiare prompt ed andare ad esempio su wt0: set sqlprompt “nuovoprompt”

**OGGETTI SQL**

**DIZIONARIO DATI**

Il dizionario dati è una collezione di tabelle che contengono le informazioni di tutti gli altri oggetti che compongono il DB.

La tabella principale si chiama DICTIONARY(DICT) questa contiene il nome e la descrizione di tutte le tabelle.

Di queste una delle tabelle più utilizzate è USER\_CATALOG(CAT) che mostra tutti i principale oggetti dell’utente connesso.

Possiamo vedere il contenuto di DICT con il comando: select \* from dict;

Il risultato sarà molto grande, quindi conviene formattare le linee in questo modo:

set line 156 (imposta il numero di linee)

col comments for a120 (imposta la larghezza dei commenti a 120)

set pages 1000 (setta la pagina a 1000 righe)

Le tabelle del dictionary sono di questo tipo:

USER\_ : contengono informazioni sugli oggetti di proprietà dell’utente

ALL\_: contengono informazioni sugli oggetti accessibili dall’utente, quindi i propri oggetti ma anche oggetti di altri utenti ai quali è stato dato un GRANT.

DBA\_ : informazioni su tutti gli oggetti database ed è accessibile solo agli amministratori

V$ e GV$: Sono viste dinamiche sullo stato del DB, quindi ad esempio quanti utenti sono connessi ora.

CDB\_ :oggetti nel db contenitore e pluggalbe DB.

**TABELLE**

In casi particolari le tabelle possono risiedere in al di fuori del database e in questo caso si chiamano External Table, oppure essere suddivise in sottotabelle(Partizioni)

La caratteristica più importante sono sicuramente le colonne, ogni colonna è definita dal nome dal tipo di dato e dalla sua obbligatorietà, l’obbligatorietà è definita come “NOT NULL”.

Il tipo di dato può essere ad esempio CHAR(20) significa che è di tipo alfanumerico con al massimo 20 caratteri.

**Contraint**

I contraint sono delle regole per verificare i dati inseriti nella tabella, uno dei contraint è appunto “NOT NULL”

-Primary key: è la chiave primaria della tabella, quindi questa colonna deve essere sempre valorizzata e deve essere univoca.

-Foreing Key: Chiave esterna, sostanzialmente una colonna di una tabella detta figlia, fa riferimento alla chiave primaria di un’altra colonna. In questo caso i dati inseriti nella chiave esterna della tabella figlia devono essere già presenti nella chiave primaria della tabella padre.

-Unique Key: In questo caso i dati di una colonna che non è chiave primaria, devono comunque essere univoci (Es. Codice fiscale)

-Check Contraint: Si impone una regola ad una colonna che deve assumere solo determinati valori, ad esempio un campo DATA deve essere maggiore di xx/yy/zzzz o che devono assumere solo Y/N

**UNDO tablespace**

Come dice la parola l’undo serve per tornare indietro da una modifica ad una tabella, ad esempio se faccio un insert di una riga, oracle memorizzerà come undo un comando inverso di delete, se modifico un valore con un UPDATE oracle memorizzerà nella UNDO l’update con il valore precedente e così via.

I record di UNDO vengono memorizzati nell’undo tablespace dove il proprietario è SYS, un record di UNDO normalmente viene mantenuto finchè l’utente non fa un commit, fichè non fa un rollback, finchè la sessione non viene chiusa normalmente o finchè la sessione non viene chiusa brutalmente(in questo caso farà un undo automatico).

L’undo/RollBack segment serve chiramente per fare un rollback, ma anche per tornare indietro da una transazione andata male oppure molto importante per il Read Consistency.

Se voglio impedire che oracle sovrascriva un record di UNDO finchè non scade la retention, posso abilitare il RETENTION GUARANTEE.

Read Consistency: Se ad esempio faccio una query alle 10:30 che richiede 20 minuti per dare un risultato, probabilmente i dati nel frattempo possono essere modificati, ma oracle grazie al Rollback segment può restituire i valori che erano presenti alle 10:30 quando è stata lanciata la query.

ORA-01555 snapshot too old Error: Sempre nell’esempio di prima, se impiega così tanto tempo è possibile che non ci sia più memoria nella rollback table oppure si è raggiunto un tempo di retentions del dato e quindi oracle non potrà restituire dati coerenti così restituisce questo errore.

Di default viene creata una UNDOTBS1 gestita automaticamente(consigliato) che si autoestende e gestisce la retention automaticamente a seconda delle query lunghe che vengono lanciate.

**REDO Management**

Al contrario della Undo table, i redo servono per ripristinare le transazioni fatte dopo un crash. Quindi quello che viene scritto sul redo è coerente con la transazione (se ho fatto una insert nel redo ci sarà scritta la insert)

Chiaramente c’è un redo log buffer, che viene scritto sul redo log file(nel disco) in 3 condizioni: Quando c’è un commit, quando il log buffer è pieno, prima di un DBwn, oppure ogni 3 secondi.

I redo log sono più di uno e vengono ciclati, normalmente si utilizzano il log group(tipo raid1) in modo che di ogni log c’è anche un’altra copia per eventuali corruzioni.

Il Current log file è quello che è attualmente scritto, gli active log file sono tutti quelli che non possono ancora essere sovrascritti da oracle perché ancora non sono stati passati in archive. Mentre il inactive log file sono quelli che possono essere sovrascritti.

Per vedere i log file: select \* from v$logfile;

per aggiungere un logfile ad un log group:

alter database add logfile member ‘/disk2/prod1/log/redo01b.log’ to group 1;

Oltre ai redo log, ci sono gli archive, quando un log in è active questo viene anche copiato negli archive.

**Gestione Database User**

Ogni utente ha le seguenti propretà: Unique Username, authentication method, default tablespace, temporary tablespace, user profile, initial consumer group, account status.

Uno schema è la collezione di tutti gli oggetti di proprietà dell’utente, lo schema si chiama sempre come l’utente.

Utenti di default: SYS(fa tutto), SYSTEM(tutto tranne Backup/Restore), DBSNMP(Enterprise manager solo per analisi), SYSMAN(Enterprise manager per gestione).

Create user john identified by password123;

GRANT CONNECT TO John;

GRANT RESOURCE TO JOHN;

**PARTIZIONI PARTITION TABLE**

Servono per dividere i dati di una tabella in partizioni per maggiori performance, resilenza, operazioni contemporanee.

**RANGE PARTITION:**

Viene partizionata la tabella con dei range impostati ad esempio, creo la tabella sales e la divido in due partizioni temporali:

create table sales (

customer\_id Number,

order\_date DATE)

PARTITION BY RANGE (order\_date)

(PARTITION sales\_P1 value less than (TO\_DATE(‘2021-06-30’, ‘YYYY-MM-DD’)),

(PARTITION sales\_P2 value less than (MAXVALUE));

Volendo si può atuomatizzare la creazione delle partizioni, aggiungendo:

INTERVAL (NUMTOMYINTERVAL(1,’MONTH’))

In questo modo ogni mese che viene aggiunto, aggiunge in automatico una partizione.

**PARTITION LIST:**

Divide la tabella utilizzando dei nomi descrittivi, ad esempio divido per nord-est e sud-ovest.

create table sales (

customer\_id Number,

Region varchar2(10))

PARTITION BY LIST (REGION)

(PARTITION sales\_R1 VALUES(‘nord’,’est’),

(PARTITION sales\_R2 VALUES(‘sud’,’ovest’);

**HASH PARTITION**

In pratica sulla colonna selezionata crea un algoritmo di hash e suddivide in maniera automatica e più o meno equa i dati nelle partizioni create. Non c’è un controllo su dove andranno a finire i dati

create table sales (

customer\_id Number,

Region varchar2(10))

Partition by hash(customer\_id)

(partition c1,

partition c2));

Le partizioni possono essere annidate, ad esempio faccio un range partition e una sub hash partition per suddividerli ancora.

Per modificare una tabella già creata aggiungendo una partizione:

ALTER TABLE SALES1 ADD PARTITION

(PARTITION 'P1550' VALUES LESS THAN TO\_DATE ('2016-01-01', 'YYYY-MM-DD'));

**INDICI**

Gli indici servono per ottimizzare una ricerca, quando creo un indice Oracle crea una sovrastruttura ordinando i dati in un certo modo per migliorarne la ricerca. Ad esempio in una anagrafica se metto l’indice per cognome, creerà delle sovrastrutture con un indice per lettera.

**Indice B-Tree**(balanced tree): è l’indice di default, prendendo l’esempio dell’anagrafica indicizzata per cognome, oracle creerà una struttura ad albero dove nell’ultimo livello ci saranno tutte “foglie” con i cognomi e su queste foglie ci sarà l’esatta posizione nella tabella (file,blocco,riga)



**Indice Bitmap**: Quando una colonna ha pochi valori univoci tipo Sesso (M,F) l’indice B-Tree è inefficace, in questo caso è meglio un indice bitmap, in questo esempio creerà due soli rami(M,F) e sulle foglie ci saranno tutti i valori della tabella con un bit(0/1) per identificare se quella determinata riga è M o F

Si possono anche creare più indici sullo stesso insieme di colonne ma non ho capito come.

**IOT**(index organized table) è una tabella organizzata in una struttura B-Tree automatica basata sulla chiave primaria. Va da se che in questo caso la chiave primaria è obbligatoria e che è adatta quando le ricerche si concentrano sulla Primary Key.

**Cluster Tables**

Se due tabelle vengono lette molto spesso insieme, è conveniente fare un cluster di tabelle. In questo modo oracle salverà righe di entrambe le tabelle nello stesso blocco di memoria fisica.

Per creare un cluster di tabelle, è necessario individuare una chiave di clusterizzazione che generalmente accomuna i valori di entrambe le tabelle e spesso fa riferimento a delle foreign key.

**Viste**

Le viste servono a semplificare la scrittura delle query SQL, infatti non sono altro che delle query su una o più tabelle alle quali si da un nome, è quindi possibile utilizzare questo nome come fosse una vera tabella per scrivere una successiva query, le viste normali non impattano su disco.

Come ho detto possono essere utilizzate come fosse una tabella e quindi ci posso scrivere dei dati che poi finiranno nelle tabelle reali, qui si può incappare in un paradosso, prendiamo una vista semplice su una anagrafica FEMMINE(vista), posso aprire questa vista e compilare una anagrafica con scritto però M, in questo caso creo un record che non può essere visto dalla vista stessa. Per ovviare a questo paradosso posso selezionare il **Check-Option\_Contraint** in questo modo oracle impedisce di inserire dati in una vista che non potrebbe vedere.

**Viste Materializzate Materialized view**

Se una vista è complessa con moltissimi dati, potrebbe essere lenta. Quindi posso fare una vista materializzata che di fatto crea una nuova tabella fisica con i risultati della vista per essere più veloce. In questo caso i dati vanno in qualche modo aggiornati sulla vista materializzata perché le tabelle originali possono cambiare, quindi posso impostare un refresh periodico di tipo completo(riscrive tutta la tabella materializzata) o Fast(tiene un log delle modifiche sulle tabelle originali e cambia solo i valori necessari).

Oppure posso fare un refresh manuale con il comando REFRESH

Le viste materializzate sono anche molto usate per fare le repliche tra server in datacenter diversi.

**OPZIONI VISTE MATERIALIZZATE**

Immediate: La vista è popolata immediatamente

Deferred: La vista verrà popolata al primo refresh

FAST Refresh: Refresh incrementale, necessita di un log altrimenti fallisce

Complete Refresh: Cancella e riscrive tutta la vista

Force: prova un FAST e se fallisce fa un Complete

On Commit: Il refresh viene triggerato ad ogni commint change delle tabelle sorgenti

On Demand: Il refresh viene triggerato manualmente o con un task sheduler.

EXEC DBMS.MVIEW.REFRESH(‘VistaMaterializzata’);

Se voglio schedulare un refresh d 1 minuto, nei comandi della creazione della vista devo aggiungere:

start with sysdate next sysdate + 1(24\*60)

Opzione “enable query rewrite”: Nella creazione della vista, se aggiungo questa opzione, oracle potrà utilizzare questa vista per delle altre query inserendola nei piani di esecuzione.

**Sequenze**

Genera dei numeri normalmente utilizzati per valorizzare una chiave primaria

**Sinonimi**

Sono degli alias, tipo CAT è sinonimo di sys.user\_catalog.

Non c’è limite al numero di sinonimi assegnati allo stesso oggetto, i sinonimi creati possono essere pubblici(utilizzabili da tutti e non necessitano di anteporre nulla) privati(ad uso del creatore, il creatore può concedere il permesso ad un altro utente e in questo caso nell’utilizzo si deve anteporre il nome dello schema proprietario ad es luca.sinonimo

**Database Link**

Collega due database in maniera unidirezionale, il DB che necessita di accedere all’altro database funge da client e utilizza una stringa di connessione.

Create database link MY\_LINK

 Connect to Pippo IDENTIFIED BY Password123

 USING ‘TNS\_ENTRY\_NAME’;

per richiamarlo: select \* from tabella@MY\_LINK

**Diagnostic Data**

-Alert log

-Trace files

-Core Dump files

ADR (automatic diagnostic repository) posiziona tutti questi file di log in: ADR BASE(di solito oracle\_home)\diag\rdbms\database name\SID\

 Per vedere la ADR\_BASE: show parameter diagnostic\_dest

In particolare gli alert log sono in formato XML, per leggere meglio i diagnostici e correlare gli eventi posso utilizzare il tool ADRCI

Una volta entrati in ADRCI posso ad esempio fare “show alert”

show alert -tail 20

show alert -p “message\_text like ‘%ORA-‘” (mostra solo i messaggi di errore)

show problem

show incident

**Backup & Recovery**

Se il database è configurato in modalità NO-ARCHIVE, in caso di failure non posso far altro che ripristinare dall’ultimo backup e non posso applicare gli online redo log.

Se invece il database è configurato in ARCHIVE-Mode, in caso di failure andrò a ripristinare il backup, poi uno ad uno i successivi archive e infine l’ultimo redo log. Non perdendo nulla

No-Archive: in questa modalità posso solo fare cold backup che necessita di mettere il DB Offline

Archive-Mode: Posso fare anche degli hot backup con DB online.

**COLD BACKUP:**

Check delle posizioni dei file:

select file\_name from dba\_data\_files;

select name from v$controlfile;

archive log list;

Poi spegno il DB, copio le varie cartelle o faccio un TAR e riaccendo il db.

**HOT BACKUP:**

1. Verificare che il db sia in archive mode (archive log list;)
2. Determinare i percorsi dei file (come per il cold backup)
3. Verificare il sequence number degli online redo log (ad es. 41)

(Select group#, sequence#, status from v$log; )

1. Settare il db in backup mode (alter database begin backup; )
2. Backup dei data file con CP o TAR
3. Toglier il backup mode dal backup (alter database end backup; )
4. Lanciare un checkpoint (alter system archive log current; )
5. Backup dei control file

(alter database backup controlfile to ‘/backuplocation/control.bk’; )

1. Verificare l’attuale sequence number (ad es 46)

(Select group#, sequence#, status from v$log; )

1. Fare il backup degli archive log del delta sequence number(41,42,43,44,45)

Li copio con CP dalla cartella di archive

**Recover dopo aver perso i control file:**

spegnere il DB, copiare il backup dei control file verso la posizione giusta e rinominarli dando il nome di quelli giusti, dare i giusti permessi a questi file. Poi fare un recovery del db:

startup mount

recover database using backup controlfile until cancel;

specificare il primo dei redo log, se non gli piace andare avanti a specificarli fino all’ultimo.

Alter database open resetlogs;

**Recover dopo corruzione data file**

Copio il file dal backup e riassegno l’owner e il gruppo

Startup mount;

recover tablespace system;

quando viene chiesto il file di log posso mettere AUTO.

ALTER DATABASE OPEN;

**Backup & Recovery con RMAN**

FULL BACKUP: Il solito full

Incrementale:

prima cosa divide in due tipi di backup, level-0 e level-1 dove level-0 è il primo backup completo che fa da padre al resto dei backup chiamati level-1.

Inoltre i level-1 possono essere:

Incrementali\differenziali: Come i classici incrementali

Cumulativi: Sarebbero i classici differenziali

Quando si utilizza RMAN, questo di default scrive molte informazioni nel control file del DB. Per Evitare questo comportamento la best practice è quella di creare un DB apposito chiamato Recovery\_Catalog

I backup con RMAN possono usare due metodi di copia:

Image-copy: Copia i file uno ad uno, copiando anche i blocchi vuoti. NON può essere fatto su TAPE, il vantaggio è che posso recuperare anche un singolo file.

Backup-Set: Inserisce tutti i file in dei set, è compresso, supporta i TAPE, non copia i blocchi vuoti, lo svantaggio è che non posso recuperare un singolo file. Ad ogni modo è il backup consigliato da oracle.

I comandi sono nel file dei comandi :D

**Oggetti PL/SQL**

È un linguaggio di programmazione procedurale che incorpora il linguaggio SQL.

Ci sono Funzioni, Procedure, package e Trigger(i trigger non possono essere richiamati ma scattano automaticamente ad una determinata condizione.

72

**Creazione di una tabella**



Posso creare una tabella clone di un'altra con:

WTO >create table test\_copy as

 select \* from test;

Normalmente quando creo una tabella vado ad utilizzare il tablespace predefinito dell’utente, ma volendo posso specificarne uno:

WTO >create table test (

A number,

B number)

 tablespace USERS;

**Creazione di un indice**

Indice di default:

create index test\_indice on test-table(ColonnaA,ColonnaB);

indice bitmap:

>create bitmap index test\_idx on test(a,b);

Creazione di un indice cluster:

create cluster my\_clust (a number);

Creazione di una Organized table IOT



**Creazione di una vista**

*create view clienti\_roma as*

 *select nome, cognome*

 *from clienti*

 *where comune='H501';*

Se vogliamo modificare una vista esistente si può utilizzare create or replace, per esempio inserendo il constrait

*Create or replace*

*view clienti\_roma as*

 *select nome, cognome*

 *from clienti*

 *where comune='H501'*

*with check option;*

Se voglio farla materializzata, basta cambiare “create materialized view.. ecc”

Per aggiornare questa vista materializzata, si esegue il comando *exec dbms\_mview.refresh('CLIENTI\_ROMA\_MAT','C')*

**Sequenze**

Per creare una sequenza, utile ad esempio in una chiave primaria

create sequence seq\_test; Crea la sequenza

select seq\_test.nextval from dual; Aumenta di un valore

select seq\_test.currval from dual; Restituisce il valore corrente

**Sinonimi**

Per creare un sinonimo basta fare

Create public schema.sinonimo for schema.tabella

create public synonym C2 for clienti;

**Database Link**

CREATE DATABASE LINK scott\_remoto //il nome del database link

CONNECT TO scott IDENTIFIED BY tiger //username e password per l’altro DB

USING 'altro\_db'; //altro\_db è la connection string precedentemente creata “tnsnames.ora”

**Alter table**

Si utilizza per modificare le proprietà di una tabella o colonna ad esempio:

alter table test add (D number, E date); //aggiunge la colonna D e E

alter table test modify (c number not null); //modifica la colonna C come not null

alter table test rename column A to New\_A; //rinomina

alter table test drop column d; //elimina la colonna

truncate table test; cancella tutti i valori della tabella test

**Colonna invisibile**

È possibile creare/modificare una colonna e impostarla come invisibile, in questo modo se faccio una semplice select \* from tabella, la colonna invisibile non verrà presa in considerazione. Per prendere i valori della colonna invisibile, va esplicitata la colonna in fase di query.

alter table test\_invisibili modify b invisible;

Constraint

Esempi:

*ALTER TABLE TEST ADD CONSTRAINT TEST\_PK PRIMARY KEY (A);* //aggiunge un constraint chiamato test\_pk il quale rende la colonna A primary key

*alter table test add*

*constraint test\_fk foreign key (c) references test2(chiave);* //Aggiunge un constraint il quale referenzia la colonna C come test2(chiave)

*alter table test add constraint test\_uk*

*unique (x);* //rende la colonna X unique

*alter table test add*

*2 constraint test\_ck check (Y>3);* //la colonna Y accetta solo valori maggiori di 3

*alter table TEST\_DUP rename constraint X\_PK to NUOVO\_X\_PK*; //rinomina un constraint

Se voglio eliminare una tabella, la quale ha una sua colonna referenziata da una foreign key in un'altra tabella, non posso farlo con il semplice DROP. Devo utilizzare la clausola cascade constraints; che eliminerà anche la constraint.

**Cestino Purge**

Quando elimino un oggetto, questo finisce nel cestino

Per vedere il contenuto del cestino:

 *select OBJECT\_NAME, original\_name, type*

*from user\_recyclebin;*

per svuotare il cestino:

*PURGE RECYCLEBIN*

Per purgare una tabella o indice

*Purge table|index nometabellaOindice*

Per recuperare un elemento dal cestino:

*FLASHBACK TABLE <nome schema>.<nome tbella> TO BEFORE DROP*

**Insert e DML**

Quando inserisco i dati ci sono delle cose da considerare:

Numeri: la virgola si esprime con il punto.

Stringa: si mette tra apici ‘, se ho un apice all’interno del testo si usa il doppio apice 'Quest''esempio'

Date nel formato: DATE'2011-01-28' o TIMESTAMP'2011-01-28 23:59:59'

Inserimento di dati semplice, in questo caso devono essere valorizzati tutti i campi ma volendo posso specificare null per non assegnare nessun valore:

insert into test

values (null, date'2011-01-13', 'Stringa d''esempio');

Se invece voglio valorizzare solo due colonne delle tre per questa tabella:

insert into test (A, B)

values (123.4, date'2011-01-13');

Qundo creo una tabella, posso specificare per una colonna un valore di default, in questo caso se non viene valorizzata assumerà questo valore.

*create table test\_default (*

*a number,*

*b varchar2(30) default 'Valore di default';*

Volendo in una tabella già popolata, posso inserire una nuova colonna con valore di default, questo avrà effetto retroattivo sui dati precedentemente inseriti in tabella.

Generare una tabella con una colonna presa da una sequenza, in questo esempio andando a popolare la colonna Descr, verrà popolata anche ID con un progressivo.

*create sequence test\_seq;*

*create table test\_seq\_tab*

*(id number default test\_seq.nextval,*

*descr varchar2(30));*

lnsert di una tabella partendo dal risultato di una query su un’altra tabella, è possibile farlo in vari modi per estrarre in ordine diverso o la sorgente o la destinazione.

insert into test2

select b, a, c from test;

OPPURE:

insert into test2 (E, D, F)

select \* from test;

**Update/Delete**

Chiaramente Update si utilizza per aggiornare i dati di una tabella:

update persone

set data\_nascita=data\_nascita+1

where nome=’Victoria’;

Stessa cosa delete:

delete from persone

where nome=’Victoria’;

**Merge**

Con il merge posso fare un update e insert nella stessa query ad esempio:

MERGE into MYEMP //fai il merge nella tabella myemp

USING EMP //utilizzando i dati della tabella EMP

ON (emp.empno=myemp.empno) //condizione per verificare se è vera farà un update, se è falsa un insert

WHEN MATCHED THEN

UPDATE SET //appunto se la condizione è vera fa un update delle varie tabelle di myemp

ename = emp.ename, //con i dati di emp.nomecolonna

job = emp.job,

mgr = emp.mgr,

hiredate = emp.hiredate,

sal = emp.sal,

comm = emp.comm,

deptno = emp.deptno

WHEN NOT MATCHED THEN //se la condizione non è soddisfatta, quel record non esiste e faccio un insert

INSERT (empno, ename, job, mgr, hiredate, sal, comm, deptno)

VALUES (emp.empno, emp.ename, emp.job, emp.mgr, emp.hiredate,

emp.sal, emp.comm, emp.deptno);

131

Gestione degli errori

Se ad esempio voglio fare un insert di una tabella partendo dai dati di un’altra tabella, potrebbe capitare che la tabella di destinazione abbia dei requisiti diversi e quindi va in errore e non inserisce nulla.

Per fare in modo che i dati buoni vengano inseriti e quelli anomali scartati, va creata una tabella di erros:

exec DBMS\_ERRLOG.CREATE\_ERROR\_LOG('Nometabella');

Quando faccio la query di insert:

insert into emp2

select \* from emp

log errors reject limit 20

137

**Select per ricercare**

Dal comando base select \* from clienti; posso specificare solo alcune colonne come select prodotto from clienti;

Posso anche dare un alias alle colonne estratte come “select prodotto as COD\_PROD from clienti;”

Quando faccio una select, potrei chiedere di inserire un valore che non è presente in tabella ad esempio:

select 3\*5 from clienti; in questo modo verrà restituito 15 per tante volte quante righe ci sono nella tabella, se invece voglio far uscire fuori una sola volta 15, posso utilizzare la tabella di sistema “dual”, che è una tabella dummy con un solo record. Quindi se faccio select 3\*5 from dual; uscirà fuori 15



**Selezioni Where**

Ad esempio:

select \* from comuni

where provincia='RM';

Gli operatori logici sono i soliti: = != > <

Questo non vale per verificare i valori null e not null, ovvero non posso fare un “where provincia = null”, devo utilizzare “where provicia Is Null” oppure “is not null”

Gli operatori si possono utilizzare anche per confrontare stringhe di caratteri e date, attenzione perché le date hanno intrinsechi anche le ore i minuti e secondi, se non vengono valorizzati di default ci sarà comunque 00:00:00

Between|not between: è un range, del tipo: where importo between 400 and 700

IN, NOT IN: se appartengono ad un elenco ad esempio “where importo in (100,200,700)

LIKE: confronta le stringhe che metchano un pattern: ad esempio “where provincia like ‘\_M%’ significa qualsiasi stringa che inizia per un qualsiasi carattere, secondo carattere M e poi qualsiasi cosa. Quindi \_ significa singolo carattere mentre % significa qualsiasi stringa di caratteri.

Si possono concatenare più condizioni con AND OR e NOT

**Ordinamento**

Esempi:

select \* from clienti

order by comune, cognome; Ordina in base al comune e se uguale in base al cognome

Di default utilizza un ordinamento crescente, se voglio cambiarlo basta mettere DESC, order by cognome DESC.

**Raggruppamenti**
se vogliamo raggruppare i risultati, la cosa più semplice è usare DISTINCT

select distinct data\_fattura, importo from fatture; la limitazione è che le colonne da estrarre devono rientrare tutte nella distinct.

Più flessibile è sicuramente la funzione “GROUP BY”, questa non si mette sbito dopo la select ma dopo l’eventuale where. Inoltre supporta le funzioni di gruppo, ovvero delle funzioni che si applicano al raggruppamento come SUM:

select data\_fattura, sum(importo) //in questo caso raggruppa per data fattura e ne somma l’importo

 from fatture

 group by data\_fattura;

le altre funzioni sono:

MIN/MAX/AVG: estrae il valore minimo/massimo/medio di un dato presente nel gruppo

COUNT: conta le righe di un gruppo

OFFSET E FETCH

Se vogliamo estrarre solo una parte delle righe possiamo usare questi due comandi:

OFFSET: consente di estrarre le righe partendo da un determinato punto

Select \* from clienti

order by Nome OFFSET 5 rows; //parte dalla quinta riga

FETCH: estrae solo una porzione che può essere un numero esatto o una percentuale:

Select \* from clienti

order by Nome FETCH first 10 rown only; //estrae solo le prime 10 righe

Select \* from clienti

order by Nome FETCH first 40 percent rows only; //estrae solo il 40%