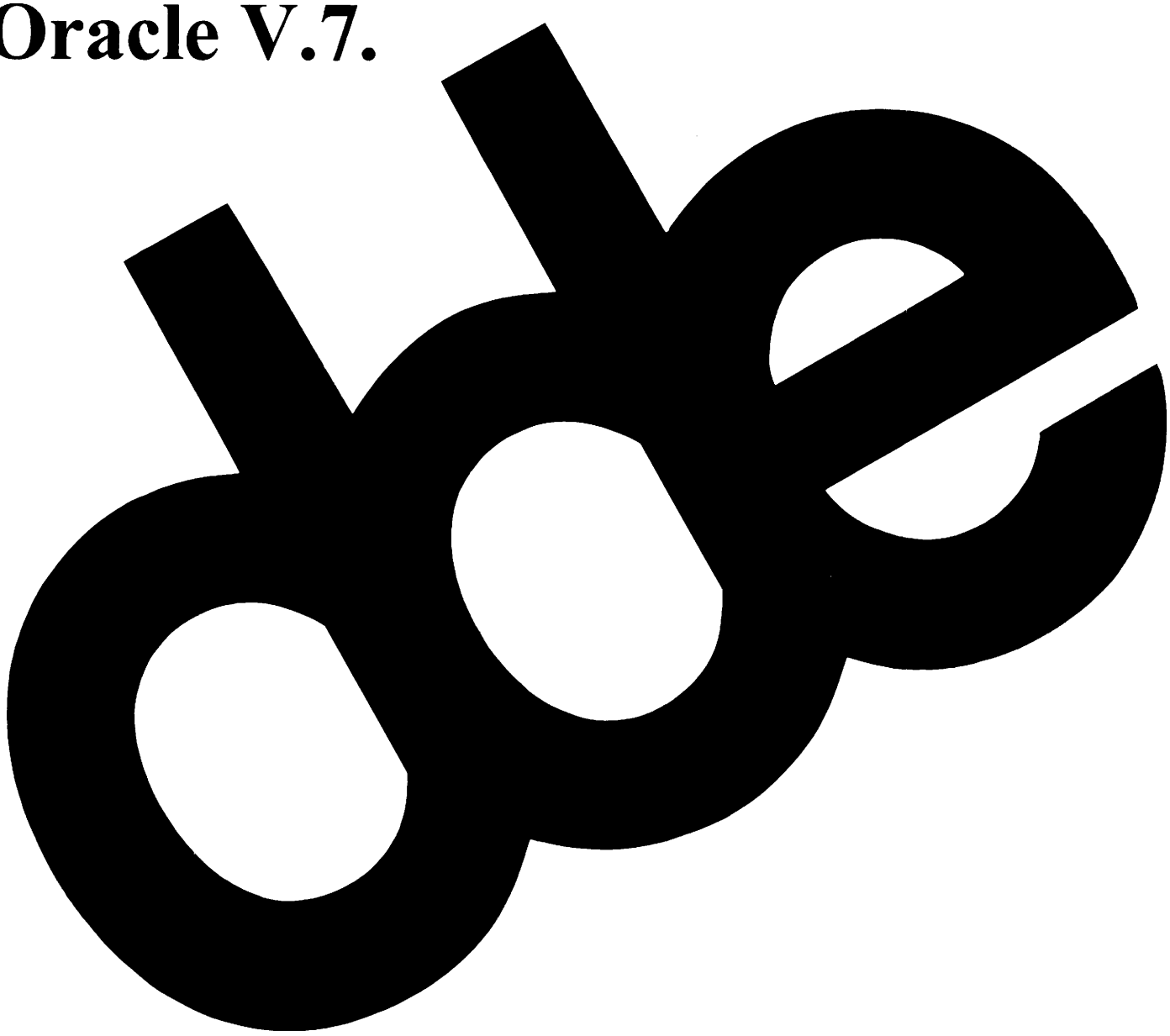
Databaseadministration 2
Oracle V.7.



August 1995

Deltagerliste :

udskrevet: 19-MAR-97

Oracle 7 Databaseadministration 2

Den 20. til 21. marts 1997

Tom Jepsen
Flyvertaktisk Kommando

Gitte Engelsholm (ge)
DDE INTERN ORDRE 131 Kursusafd.

Bernt Engstrøm, Kjær & Kjær World Wide A/S, Grønnemosevej 1, 5700 Svendborg
PROSA

Jamal Bensaoula (jab)
DDE INTERN ORDRE 131 Kursusafd.

Bogi Moutitsen (bkm)
DDE INTERN ORDRE 131 Kursusafd.

Frank Jonsson (frj)
DDE INTERN ORDRE 131 Kursusafd.

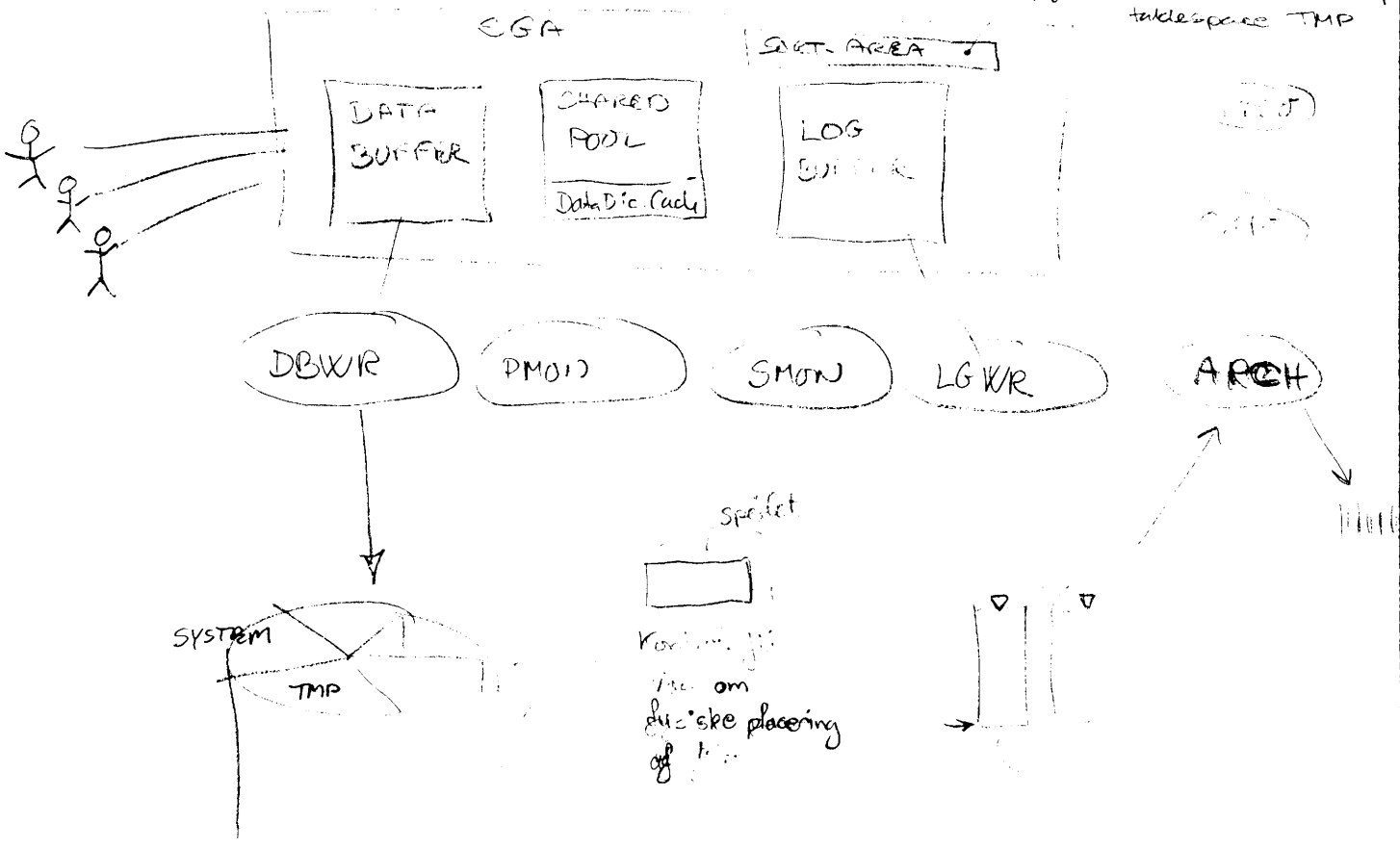
Indholdsfortegnelse

Ora 6 → Ora 7 ude af [illegible] ca. 15-25% [illegible]

V$rowcache disse [illegible] svarer til de parametre der skal konfigureres hvis man vælger at have DataDictionary adskilt fra Shared Pool.

$ORACLE_HOME/dbs/sgadefXXX.dbf Ulempen ved adskillelse er at for at huske Oracle dataene lagres derved skal der blive skrevet mere [illegible]

Når ikke stort nok [illegible] på tablespace TMP



Aktuel [illegible] værdier på [illegible] parameter

COLDER: Select * from V$parameter

sort_area_size 3 65536

DDE INTERN ORDRE
131 Kursusafd.
Att:
Herlev Hovedgade 199
2730 Herlev

Herlev, den 07. mar. 1997

Vedr.: Deltagelse i kursus

Hermed bekræftes kursusdeltagelse for:

Kursist : Gitte Engelsholm (ge)
Kursus : Oracle 7 Databaseadministration 2
Dato : Den 20. til 21. marts 1997

Kurset afholdes i DDEs undervisningslokaler på Herlev Hovedgade 199, Herlev i tidsrummet fra kl. 9.00 til 16.00

Vedlagt en folder med praktiske oplysninger vedr. kurset.

Vi glæder os til at se dig.

Med venlig hilsen
Dansk Data Elektronik A/S

Helle Møller Jensen
Kursuskoordinator

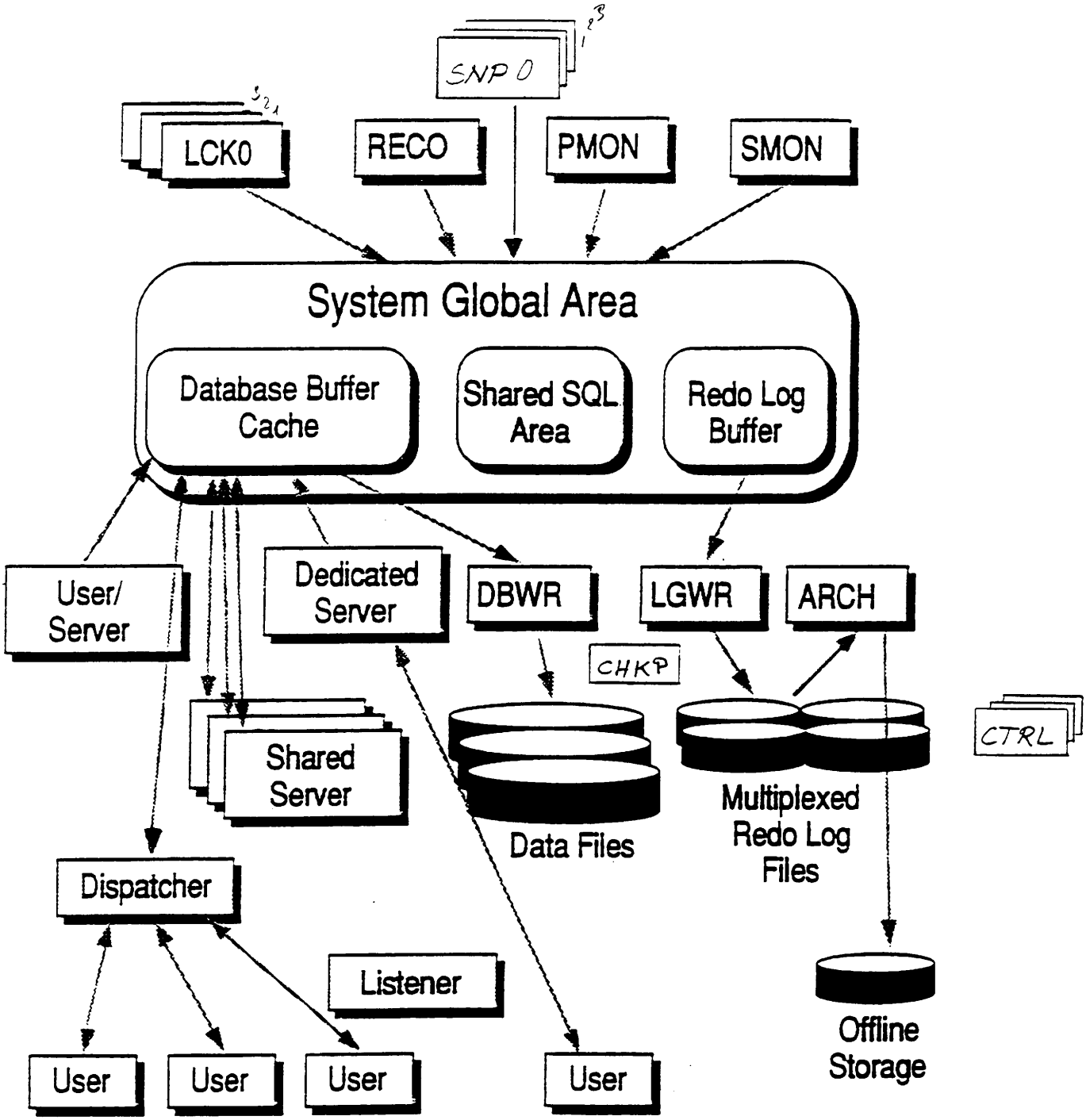
Afmeldingsregler:

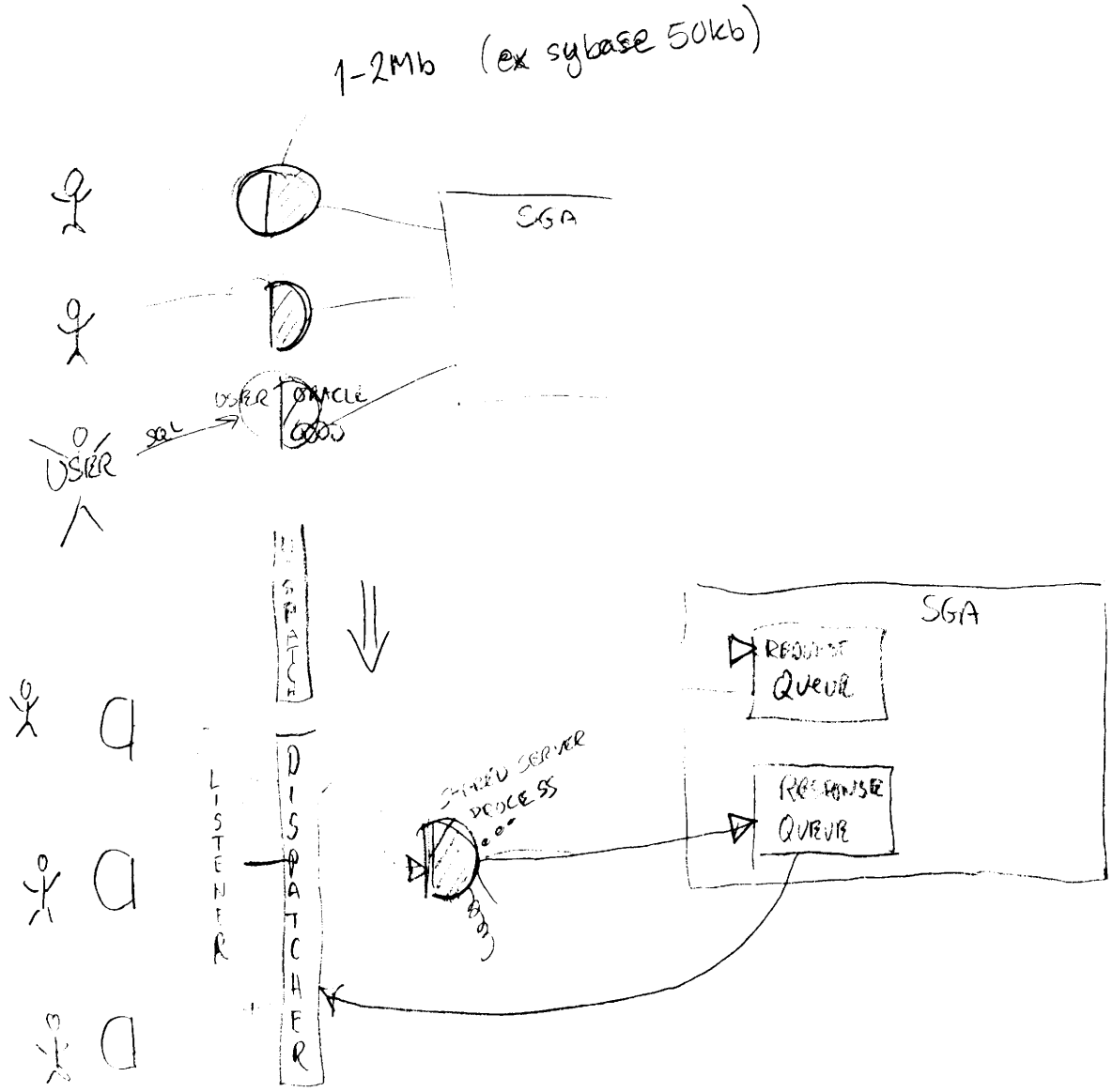
Ved afmelding senere end 14 dage før første kursusdag betales fuldt deltagergebyr. Ændring af deltager sker uden beregning.

Betalingsbetingelser:

14 dage fra fakturadato, herefter 2% i rente pr. påbegyndt måned.

1. Arkitekturoversigt V.7





Giver performance tab, men lage besparelse

2. Multi-threaded Server

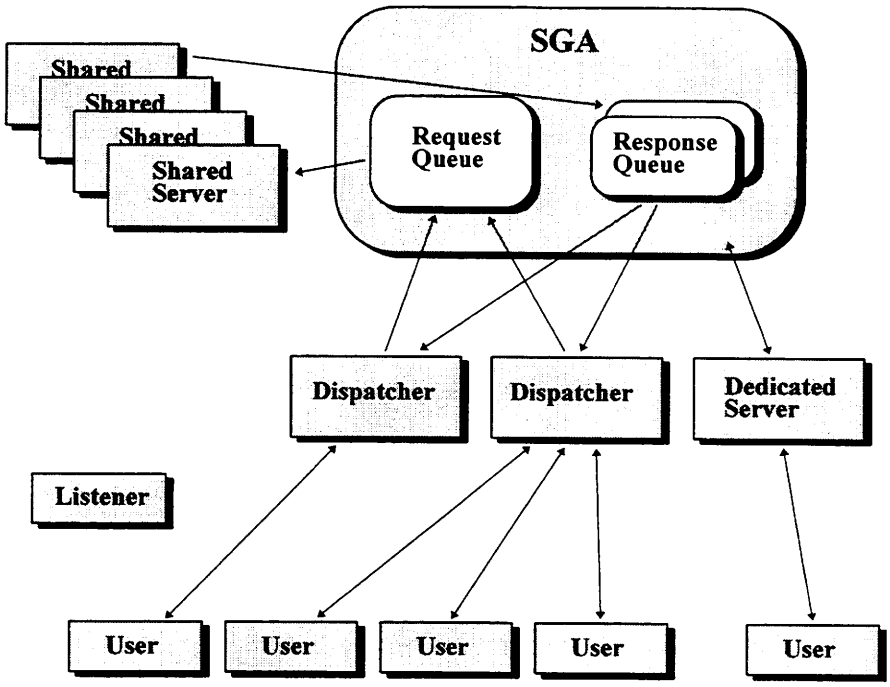
Multi-threaded Server er et nyt konfigurationsvalg, som understøtter brugerne med færre operativsystemprocesser. Dedikerede serverprocesser er, afhængig af brugernes arbejdsopgaver, inaktive i større eller mindre omfang. Idéen i en MTS-konfiguration er, at brugerne kan deles om et antal fælles serverprocesser. Det betyder en bedre udnyttelse af de startede processer.

Processer i MTS-konfiguration

Følgende processer er nødvendige i en MTS-konfiguration:

- En 'netværkslytteproces' som connecter brugerprocesser til dispatchere og dedikerede serverprocesser. Dette er en del af SQL*Net, ikke Oracle.
- Én eller flere dispatcherprocesser.
- Én eller flere fælles serverprocesser.

Bemærk, at det er nøvendigt at connecte gennem SQL*Net for at bruge fælles serverprocesser også selvom brugerprocessen findes på samme maskine som databasen.



Tnslsnr-processen

'Netværkslytteprocessen' tnslsnr kan startes vha. lsnrctl-programmet med optionen start. Tnslsnr kan kun startes af den bruger, der ejer databasen.

Eksempel

```
#  su - ora7
$  lsnrctl
LSNRCTL>  set password jlt
LSNRCTL>  start                                    eller 'status', 'help'/'start'

Connecting to (ADDRESS=(PROTOCOL=TCP)(HOST=152.95.17.101)(PORT=1521))
STATUS of the LISTENER
Alias                     LISTENER
Version                   TNSLSNR for SUPERMAX SMOS V: Version 2.0.15.0.2
Uction
Start Date                24-NOV-94      10:58:35
Uptime                    0 days   0 hr.    1 min    29 sec
Trace Level               off
Security                  ON
Listener Parameter File   /usr/ora7/network/admin/listener.ora
Listener Log File         /usr/ora7/network/log/listener.log
Services Summary ...
MTS has 1 service handles
The command completed successfully
```

Konfiguration

Konfigurationen af MTS sker dels ved hjælp af programmet 'net_conf', som på UNIX-maskiner placeres i kataloget $ORACLE_HOME/network/config/sql, og dels ved at tilrette databasens parameterfil.

Net_conf-programmet starter en form, hvori de enkelte parametre kan angives. Formen er oprettet i SQL*Forms Version 3 og gør brug af funktionstaster. En oversigt over funktionstaster kan fås ved at trykke [HOME].

Oplysninger om de enkelte skærmbilleder kan findes i SQL*Net Administrator's Guide Version 2.0 Kap. 5.

Net_conf-programmet danner på baggrund af de indtastede oplysninger en række parameterfiler. En af filerne er 'listener.ora'. På næste side vises et eksempel på indholdet af en sådan parameterfil.

De øvrige filer er 'sqlnet.ora', 'tnsnames.ora' og 'tnsnav.ora'. Alle filerne, inkl. 'listener.ora.' skal kopieres til kataloget $ORACLE_HOME/network/admin.

Dynamisk tuning af dispatchers (antal), idet den måler på V$MTS... ?) wait-states tiden i req/resp queue for at se om der skal startes flere dispatcher!

Eksempel

```
############
# FILENAME: listener.ora
# TIME....: 94-11-23 04:00:37
# NETWORK.: JLT
# NODE....: HUGO
# SERVICE.: LISTENER
############
LISTENER =
  (ADDRESS_LIST =
        (ADDRESS =
          (PROTOCOL = TCP)
          (HOST = 152.95.17.101)
          (PORT = 1521)
        ) )
STARTUP_WAIT_TIME_LISTENER = 0
CONNECT_TIMEOUT_LISTENER = 10
SID_LIST_LISTENER =
  (SID_LIST =
    (SID_DESC =
      (SID_NAME = MTS)
      (ORACLE_HOME=/usr/ora7)
    ) )
PASSWORDS_LISTENER = (jlt)
TRACE_LEVEL_LISTENER = OFF
```

eller UNV1

DECNET SERVER ...

Initialiserings parametre

I init.ora-filen findes en række parametre, der skal angives i forbindelse med MTS.

MTS_SERVERS	Default 0
MTS_MAX_SERVERS	Default 20
MTS_DISPATCHERS	Default null
MTS_MAX_DISPATCHERS	Default 5
MTS_LISTENER_ADDRESS	Default null
MTS_SERVICE	Default DB_NAME

Eksempel

```
mts_dispatchers="tcp,1"
mts_max_dispatchers=10
mts_servers=1
mts_max_servers=10
mts_service=MTS
mts_listener_address="(ADDRESS = (PROTOCOL = TCP)
                        (HOST = 152.95.17.101) (PORT= 1521))"
```

Se også sqlnet :
ORACLE_HOME/network/admin/
listener.ora

Antallet af servere og dispatchers kan ændres med kommandoen 'alter system'.

Eksempel

```
SQL> ALTER SYSTEM SET mts_servers = 10;
SQL> ALTER SYSTEM SET mts_dispatchers = 'tcp,5';
```

Connect via SQL*Net

For at connecte via SQL*Net skal evironmentvariablen TWO_TASK angives. TWO_TASK sættes til værdien angivet i filen 'tnsnames.ora'.

Kontrol af servere

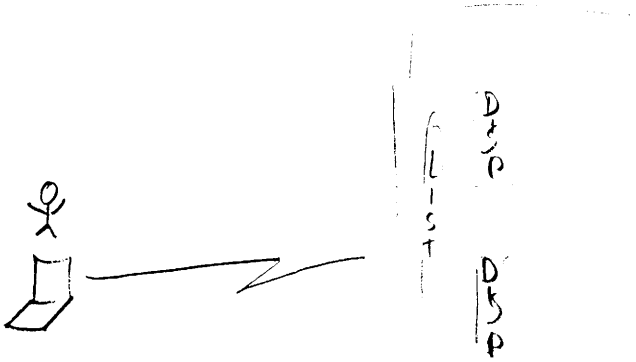
Hvis MTS er konfigureret, og der connectes via SQL*Net, skal der ikke længere findes nogen skyggeprocess. I stedet skal en eller flere fælles servere være startet op.

Om der findes en skyggeproces kan undersøges således:

```
SQL> !ps -u <UNIX-login> ^grep Oracle
```

Om der findes kørende 'shared servers' kan undersøges således:

```
SQL> SELECT * FROM v$shared_server;
```



tnsnames.ora

B3
↑
ORACLE_SID

$ORACLE_HOME/network/admin
listener.ora

B3

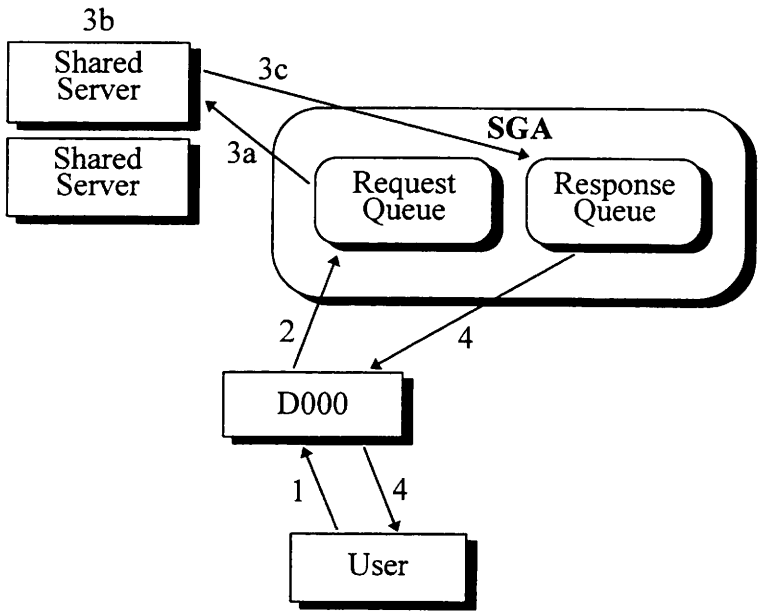
init.ora :

mts_services = B3

Hjælpeværktøjer til opsætning af listener.ora og tnsnames.ora (ikke mts. endnu i init.ora
/usr/ora7/network/config/sql/ NC_ . . . . . frm

Håndtering af anmodning

1. Brugeren sender en anmodning (request) til sin dispatcher.
2. Dispatcheren anbringer anmodnningen i 'request queue' i SGA'en.
3. Den første ledige serverproces:
   a. tager den første anmodning fra 'request queue'
   b. behandler den
   c. returnerer et svar til 'response queue' i SGA'en
4. Dispatcheren videregiver svaret fra 'response queue' til brugeren



Åbning UNIX-client til at gå ud på nettet ved at sætte

Øvelse 2.1

Formål: *at opsætte og køre på en MTS-konfiguration*

1. Undersøg om lytteprocessen 'tnslsnr' er kørende. Få i givet fald vist en status på processen
2. Undersøg indholdet af filerne i kataloget $ORACLE_HOME/network/admin.

3. Ret initialiseringsparametrene for den tildelte database, så den startes med MTS.

4. Log på basen direkte.
   a) Har du startet en skyggeproces?
   b) Er der kørende fællesprocesser?

5. Log på basen via SQL*Net.
   a) Har du startet en skyggeproces?
   b) Er der kørende fællesprocesser

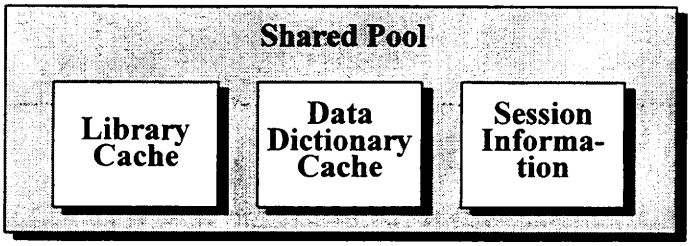
6. Start monitor programmet på en skærm tilknyttet den tildelte database. Vælg 'Multi Threaded Server Shared Server Monitor'.

7. Fra en anden skærm skal I forsøge at belaste databasen mest muligt ved hjælp af seletc-,update-, insert- og deletekommandoer.

3. Shared Pool

I SGA'en findes området 'Shared Pool', der indeholder:

- Shared SQL Areas
- Data Dictionary Cache
- Session Information



Library Cache

Library cache indeholder 'shared SQL areas' og 'shared PL/SQL areas'. Oracle repræsenterer hvert eksekveret statement med en fælles del (shared SQL area) og en privat del (privat SQL area). Oracle opdager, hvis flere brugere eksekverer det samme SQL-statement og genbruger den fælles del for disse brugere.

Shared SQL Area

Et 'shared SQL area' indeholder:

- En parsed udgave af det enkelte SQL-statement
- En udførelsesplan

Et 'shared SQL area' placeres altid i 'shared pool' og er fælles for identiske SQL-statements. SQL-statements er identiske når:

- teksten er identisk
- de er identiske mht. små og store bogstaver
- de er identiske mht. mellemrum mellem tekst
- de spørger mod samme bruger

Eksempel

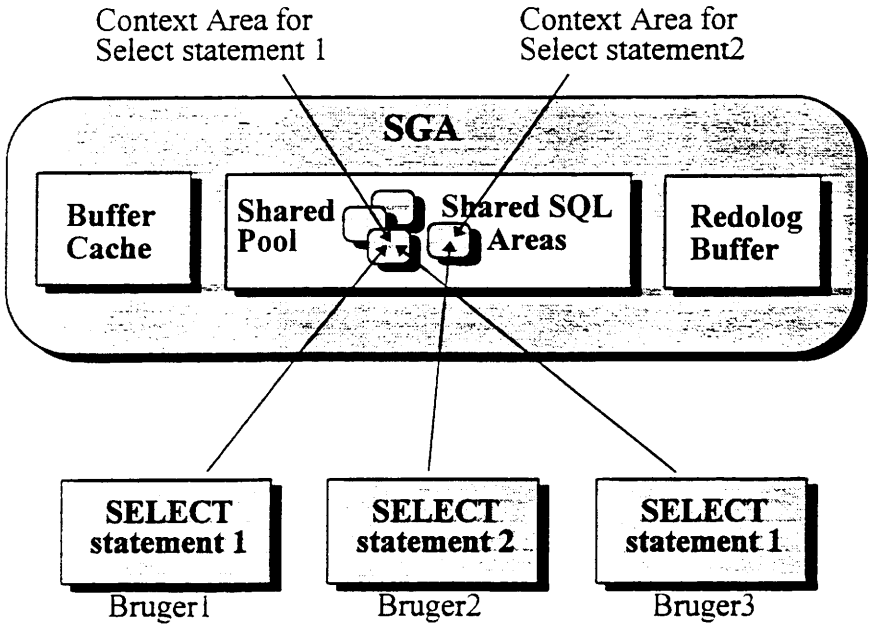
SQL> SELECT ename,job,sal FROM scott.emp

er IKKE identisk med

SQL> SELECT ename,job,sal FROM Scott.emp;

Størrelsen i bytes af Shared Pool, sættes med initialiseringsparameteren:

SHARED_POOL_SIZE	Small	3,5 Mb
	Medium	6,0 Mb
	Large	9,0 Mb



Aktivitet i Library Cache

Hvis en applikation udfører et parse-kald efter en SQL-sætning, og den parsede repræsentation ikke allerede eksisterer i et Shared SQL Area, vil Oracle parse sætningen og allokere et Shared SQL Area. Der er da tale om et Library Cache Miss.

Hvis en applikation udfører et execute-kald efter en SQL-sætning, og det Shared SQL Area, som indeholder den parsede repræsentation af sætningen, er deallokeret fra Library Cache for at gøre plads for en anden sætning, udfører Oracle en genparsing af sætningen, allokerer et nyt Shared SQL Area og udfører den. Der er også her tale om et Library Cache Miss.

For at bestemme raten af genparsing (reloads) i library cache kan følgende SQL-sætning udføres:

```
SELECT sum(pins) "Executions",
sum(reloads) "Cache Misses while Executing",
sum(reloads)/sum(pins) "Reloadrate"
FROM   v$librarycache
WHERE  namespace in ('SQL AREA','TABLE/PROCEDURE','BODY','TRIGGER')

Executions Cache Misses while Executing Reloadrate
---------- ---------------------------- ----------
      3306                            5 .001512402
```

Hvis reload-raten er større end 1%, er det et udtryk for at størrelsen af Shared Pool er for lille.

v$ sqlarea select sql_text from v$sqlarea

alter system flush shared_pool;

Systemets egen sharable area :1 :2

Øvelse 3.1

Formål: *at se indholdet af SQL Area*
at tømme SQL Area
at bestemme reload-raten

1. Udfør nogle SQL-statements.
2. Lokalisér de udførte statements gennem viewet v$sqlarea.
3. Find reload-raten i Library Cache.
4. Tøm SQL Area med følgende statement:
   SQL> ALTER SYSTEM FLUSH SHARED_POOL;
5. Undersøg indholdet af SQL Area.
6. Udfør følgende statements:
   1. SELECT * FROM scott.dept
   2. SELECT * FROM scott.dept
   3. SELECT * FROM scott.dept
   4. SELECT * FROM SCOTT.DEPT
7. Undersøg hvilke af de udførte statements der ligger i SQL Area.

sql > set timing on
sql > select sql_text from V$sqlarea

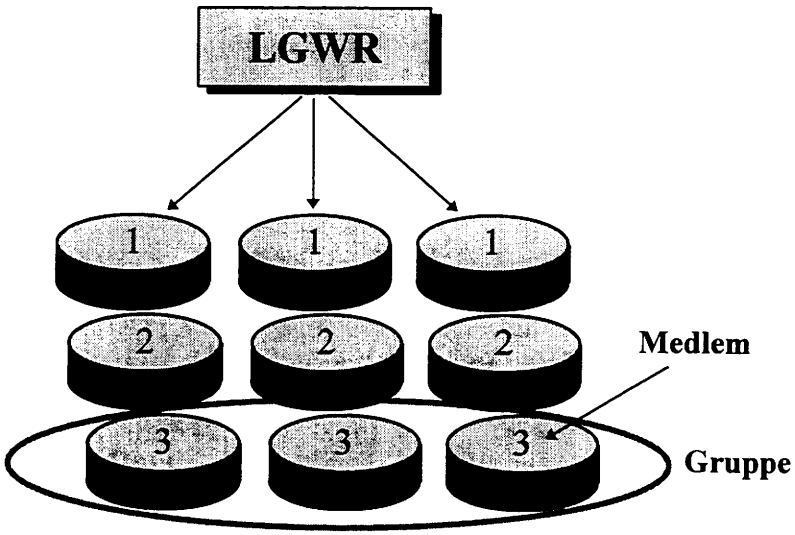
real
user
sys

4. Spejlede redologs

Overordnede betragtninger

For at modvirke at en enkelt diskfejl stopper LGWR-processen og dermed hindrer at ændringer committes, anbefales følgende:

- at hver redolog består af spejlede grupper
- at medlemmer af samme gruppe skal lagres på separate diske



Oracle tillader at spejlede grupper indeholder et forskelligt antal medlemmer.Det bør dog kun være en midlertidig tilstand. Situationen kan opstå, hvis eksempelvis en diskfejl hindrer LGWR adgang til et medlem.

Størrelse af redologfiler

I spejlede grupper skal alle medlemmer i samme gruppe have samme størrelse.

Antal redologfiler

Den optimale konfiguration er den, der har færrest mulige redologgrupper, uden at det hæmmer LGWR's skrivning af redoinformation. Det der kan ske, hvis der er for få grupper, er at en gruppe endnu ikke er tilgængelig for LGWR, når den skal genbruges. Det kan fx skyldes, at et checkpoint ikke er fuldført.

Undersøg databasens alert-fil og LGWR's trace-fil. Hvis der er meddelelser, der indikerer at LGWR må vente, så tilføj en ny gruppe.

Parametre

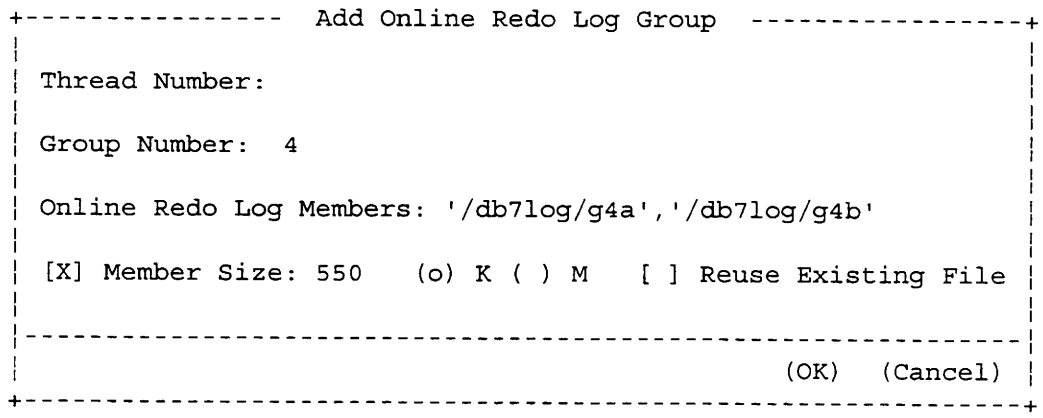
- MAXLOGFILES — Sættes som parameter i CREATE DATABASE-kommandoen. Bestemmer det maksimale antal redologgrupper, der kan oprettes. Kan undertiden aflæses i $ORACLE_HOME/dbs/crdb<sid>.ora
- MAXLOGMEMBER — Sættes som parameter i CREATE DATABASE-kommandoen. Bestemmer det maksimale antal medlemmer i en gruppe.
- LOG_FILES — Initialiseringsparameter. Bestemmer det maksimale antal redologgrupper. Værdien skal ligge i intervallet 1 - MAXLOGFILES.

System-privilegier

For at oprette nye grupper kræves systemprivilegiet ALTER DATABASE.

Opret ny gruppe

Når en ny gruppe skal oprettes, står valget mellem værktøjerne SQLDBA og SQL*Plus. Anvendes SQLDBA vælges menuen Log/Add Group.



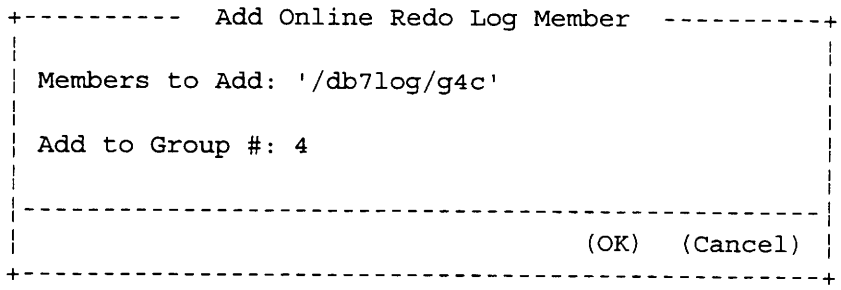
Den tilsvarende SQL-sætning ser således ud:

```
SQL> ALTER DATABASE
     ADD LOGFILE GROUP 4 ('/db7log/g4a','/db7log/g4b') SIZE 550K;
```

Hvis medlemmer ikke angives med fuld sti, anbringes de i kataloget $ORACLE_HOME/dbs.

Tilføj medlem

Tilføjelse af medlem til eksisterende gruppe udføres således gennem SQLDBA: vælg menuen Log/Add Member.

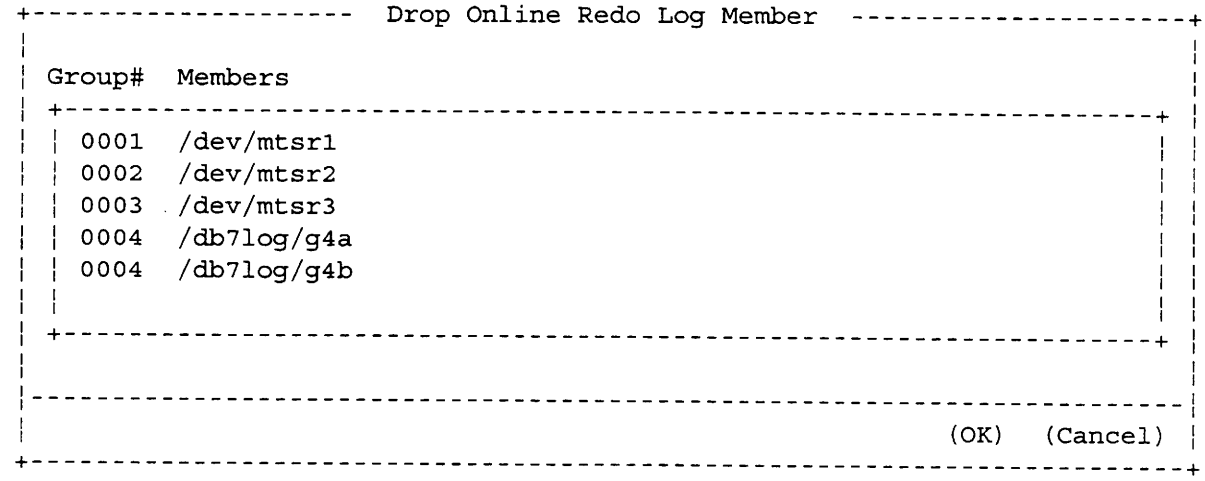


Den tilsvarende SQL-sætning ser således ud:

```
SQL>  ALTER DATABASE
      ADD LOGFILE MEMBER '/db7log/g4c' TO GROUP 4;
```

Slet medlem

Et medlem slettes fra en gruppe på følgende måde gennem SQLDBA: vælg menuen Log/Drop Member.



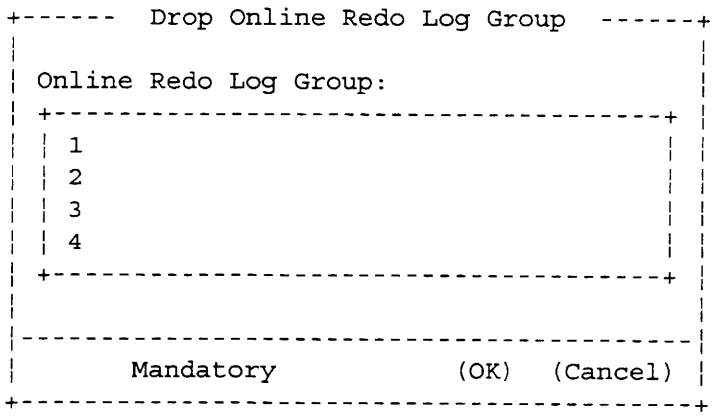
Den tilsvarende SQL-sætning ser således ud:

```
SQL>  ALTER DATABASE
      DROP LOGFILE MEMBER '/db7log/g4b';
```

Et medlem kan kun droppes, hvis gruppen ikke er aktiv, og hvis det medlem, der ønskes slettet, ikke er det sidste valide medlem i gruppen.

Slet gruppe

En hel redologgruppe slettes fra SQLDBA ved at kalde menuen Log/Drop Group.



Den tilsvarende SQL-sætning ser således ud:

```
SQL>  ALTER DATABASE
      DROP LOGFILE GROUP 4;
```

En gruppe kan ikke slettes, hvis den er aktiv. Ønskes et logswitch, udføres det med følgende kommando:

```
SQL>  ALTER SYSTEM
      SWITCH LOGFILE;
```

Når en gruppe slettes fra databasen, bliver operativsystemfilerne ikke slettet fra disken. Det er kun kontrolfilerne, der opdateres til at droppe gruppen fra databasestrukturen. Filerne skal slettes manuelt.

Information om redolog

Informationer om redologgrupper og medlemmer fås ved at spørge mod v$-tabellerne V$LOG og V$LOGFIL.

Eksempler

```
SQL>  SELECT group#,sequence#,bytes/1024 kb,members,status
      FROM   v$log;

    GROUP#  SEQUENCE#         KB    MEMBERS STATUS
---------- ---------- ---------- ---------- --------
         1          4       5112          1 INACTIVE
         2          6       5112          1 CURRENT
         3          3       5112          1 INACTIVE
         4          5        550          2 INACTIVE
```

```
1* SELECT * FROM v$logfile

    GROUP# STATUS  MEMBER
---------- ------- ----------------
         1 STALE   /dev/mtsr1
         2         /dev/mtsr2
         3 STALE   /dev/mtsr3
         4 STALE   /db7log/g4a
         4 STALE   /db7log/g4b
```

"Stale" (

"invalid"

"deleted"

sqldba ORACLE_SID = GRIP export ORACLE_SID
Connect internal

Startup ; => start

Øvelse 4.1

Formål: *at tilføje og slette grupper og medlemmer*

1. Bestem antallet af nuværende grupper samt antallet af medlemmer i dem.
2. Bestem størrelsen af de enkelte medlemmer i Kb.
3. Tilføj et nyt medlem til hver redologgruppe.
   Redologfilerne skal ligge som filer i kataloget /db7log.
4. Tilføj en ny gruppe med to medlemmer. Medlemmerne skal have samme størrelse som medlemmerne i de øvrige grupper. Filerne skal stadig placeres i kataloget /db7log.
5. Slet et medlem af den nyoprettede gruppe.
6. Prøv at slette det andet medlem af samme gruppe.
7. Slet hele den nye gruppe.

GROUP | SEQ | ARC | STATUS
3 | 6 | YES | UNUSED
1 | 23 | NO | INACTIVE
2 | 24 | NO | CURRENT

SELECT * FROM VS DATABASE

NAME | LOG MODE | CHECKPOINT CHANGE# | ARCHIVE C...
GR1B | NO ARCHIVELOG | 4015 | 3812

5. Rollback-segmenter

En Oracle 7 database oprettet med installationsscriptet indeholder 4 rollbacksegmenter. Et rollbacksegment opsamler information om aktiviteter i en transaktion. Informationen i et rollbacksegment består bla. af blokinformation på de ændrede data, samt data som de så ud inden transaktionen. Rollbackinformation om de enkelte aktioner i samme transaktion linkes sammen.

Rollbacksegmenterne kan ikke accesseres eller læses af nogen databasebruger. Kun Oracle kan læse fra og skrive til rollbacksegmenterne. Rollbacksegmenter er ejet af brugeren sys uanset hvilken bruger, der opretter dem.

Da rollbackindgange ændrer i datablokkene, bliver de også registreret i redologgen. Det er en medvirkende årsag til, at 'instance recoverery' kan håndere aktive transaktioner efter nedbrud.

Rollback segmenterne bruges til:

- at sikre læsekonsistens
- at føre transaktioner tilbage
- genetablerering af databasen.

Læsekonsistens

Oracle vedligeholder en transaktionstabel for hvert rollbacksegment. Hver tabel er en liste over alle transaktioner og tilkyttede rollbackindgange - en for hver ændring transaktionen har udført.

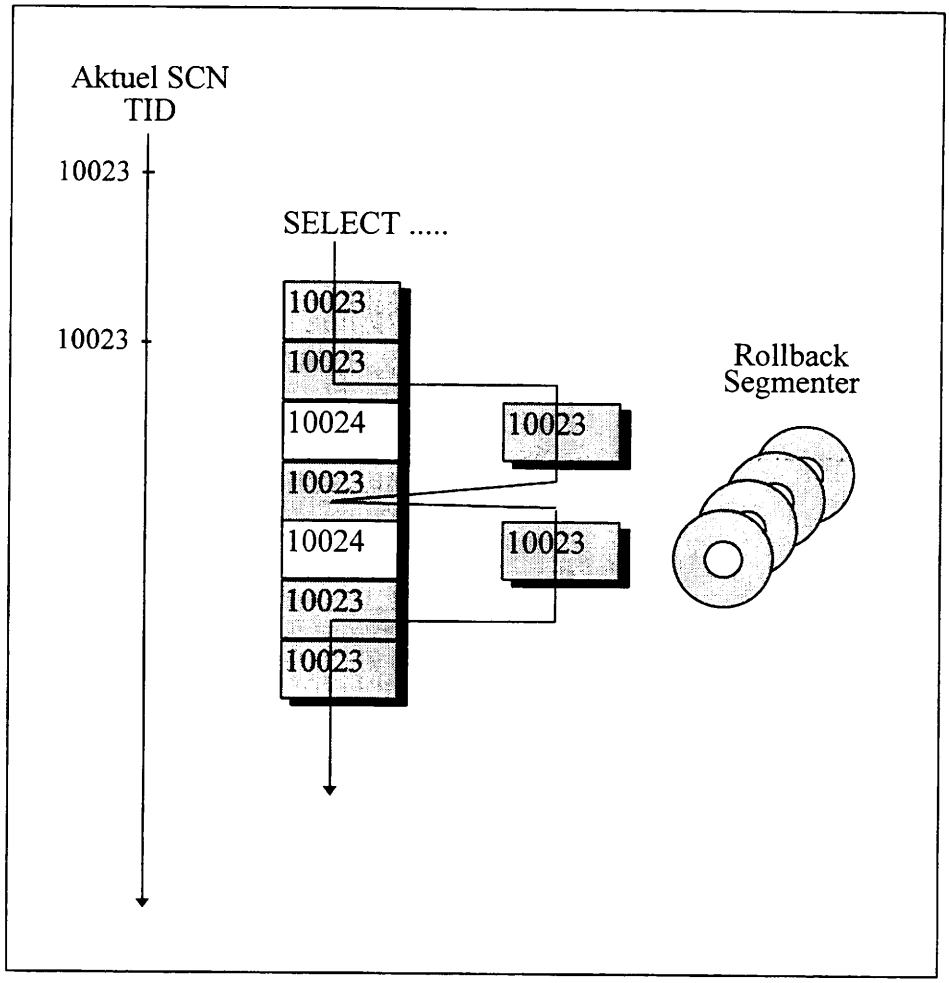
Når Oracle har brug for at producere et sæt læsekonsistente data til en forespørgsel, kan informationerne i rollbacksegmentet bruges til at oprette et sådant sæt udfra forespørgslens starttidspunkt. På det tidspunkt hvor forespørgslen går ind i execute-fasen, aflæser Oracle det aktuelle 'System Change Number' (SCN).

Når datablokkene læses for at vise resultatet af en forespørgsel, vil kun blokke med samme SCN blive anvendt. Blokke, hvori data er ændret, vil blive genskabt ud fra informationerne i rollbacksegmentet, og data herfra sendt tilbage til forespørgslen. Se figuren næste side.

Start rollback segments

cd / /init.ora

rollback_segments = (r01, r02, r03, r04)
transactions_per_rollback_segment
transactions = 20



Snapshot Too Old

Det kan ske, at Oracle ikke er i stand til at genskabe et konsistent sæt af data, også kaldet et Snapshot, til en forespørgsel, der kører længe. Den situation opstår, de når der ikke er tilstrækkelig information i rollbacksegmenterne til at genskabe de 'gamle' data. Det resulterer i fejlmeddelelsen: 'Snapshot Too Old (Rollback Segment too small)'

Én mulig løsning er, som fejlmeddelesen også antyder, at oprette større eller flere rollbacksegmenter.

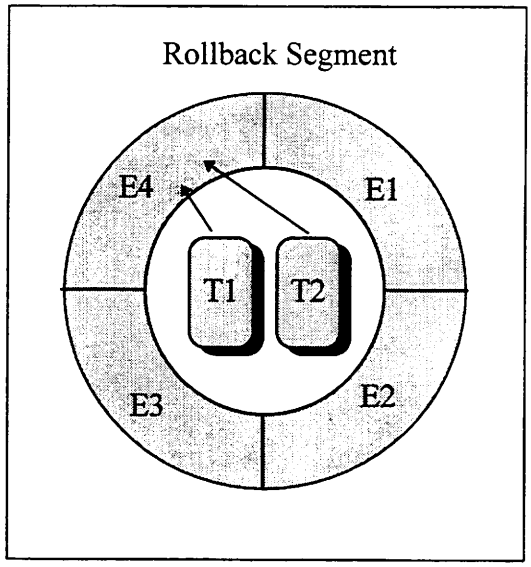
Transaktioner og rollbacksegmenter

Hver gang en bruger påbegynder en transaktion, bliver transaktionen tilknyttet et rollbacksegment. Enten gøres dette automatisk af Oracle ved den føste DML/DDL-kommando i transaktionen, eller også gives kommandoen 'Set Transaction use Rollback Segment <rbs>'.

Der skrives til det samme rollbacksegment under hele transaktionen. Når en transaktion committes, bliver rollbackinformationen frigivet men ikke slettet. Den bruges til at sikre læsekonsistente selects. For at sikre at rollbackdata er tilgængelige så længe som muligt, skrives der sekventielt til rollbacksegmentets extents. Når det sidste extent i rollbacksegmentet er fyldt, begynder Oracle at fylde det første extent, hvis det ikke indeholder aktiv rollbackinformation.

Extents i et rollbacksegment

Når et extent i rollbacksegmenetet løber fuld, vil Oracle altid prøve at genbruge næste extent i ringen først. Hvis det næste extent indeholder aktive data, må Oracle allokere et nyt extent.



Deallokering af extents

Når et rollbacksegment kreeres eller ændres, kan parameteren OPTIMAL benyttes. Optimal angiver den optimale størrelse i bytes. Optimal benyttes, når en transaktion skal fortsætte sin skrivning af rollbackinformation til et andet extent i rollbacksegmentet. Oracle sammenligner rolbacksegmentets aktuelle størrelse med den optimale størrelse. Hvis rollbacksegmentet er større end sin optimale størrelse, og det umiddelbart følgende extent er inaktivt, deallokerer Oracle sammenhængende ikke aktive extents, indtil segmentet har den mindst mulige størrelse, der er større end eller lig med den optimale størrelse.

Oracle frigiver altid de ældste inaktive extents, da det er mindst sandsynligt at de skal bruges til konsistente læsninger.

OPTIMAL kan ikke sættes til en værdi, der er mindre en rollbacksegmentet havde ved oprettelsen. Dvs.:

- INITIAL +NEXT+NEXT+ ... +optil MINEXTENTS

Når et rollbacksegment droppes, vil alle extents blive returneret til segmentets tablespace.

Eksempler

```
SQL>  CREATE ROLLBACK SEGMENT rbs02
      TABLESPACE rollback
      STORAGE(
            INITIAL         500K
            NEXT            500K
            MINEXTENTS       10
            OPTIMAL        5000K
             );

SQL>  ALTER ROLLBACK SEGMENT rbs02
      STORAGE(OPTIMAL 6000K);
```

Rollbacksegment Online/Offline

Rolbacksegmenter kan i Oracle V7 sættes online / offline med følgende kommandoer

```
SQL>  ALTER ROLLBACK SEGMENT rbs02 ONLINE;
SQL>  ALTER ROLLBACK SEGMENT rbs02 OFFLINE;
```

Øvelse 5.1

Formål: *at oprette et rollbacksegment med en optimal størrelse*
at afprøve virkningen af parameteren OPTIMAL

1. Hvilke rollbacksegmenter findes på databasen?

2. Opret et nyt rollbacksegment RBS_01, med følgende storageværdier:
   - INITIAL 10K
   - NEXT 10K
   - OPTIMAL 30K

3. Bring det nye rollbacksegment ONLINE.

4. Kopiér scott's EMP-tabel til en ora-bruger.

5. Sørg for at nedenstående transaktion udføres på det nye rollbacksegment RBS_01.

6. Udfør følgende statement indtil rollbacksegmentet overstiger sin optimale størrelse:

   ```
   SQL> INSERT INTO emp
        SELECT * FROM emp;
   ```

   Brug V$ROLLSTAT

7. Fortryd transaktionen med ROLLBACK

8. Sørg for at nedenstående transaktion udføres på det nye rollbacksegment RBS_01.

9. Udfør følgende statement medens der løbende holdes øje med V$ROLLSTAT:

   ```
   SQL> INSERT INTO emp
        SELECT * FROM emp;
   ```

10. Spørg også mod V$TRANSACTION og V$ROLLNAME. Kontrollér at det rigtige rollbacksegment anvendes.

11. Afslut transaktionen med COMMIT;

12. Slet rollbacksegmentet RBS_01.

i init NJM.ora :

control_file= /... /ctrlNJM.dbf
db_name = NJM
db_block_buffers = 700
db_block_size = 2048
shared_pool_size = 10000000

log_archive_start = true
" " format = log%s.arc
" " dest = /usr/ora7/dbs/arch

(svrmgrdba) alter database close
alter database archivelog
alter database open

? this may take a while because shutdown. Først ved 'start' genlæses ændret initNJM.ora-fil

---

rm db_data1_01

[illegible] : tabel cc switch log

Shutdown
Startup ⇒ [illegible]

cp backup db_data1_01

start ⇒ file ? [illegible]

startup mount → instance og kontrol fil
recover

6. Constraints

Det er vigtigt, at data overholder de regler, der opstilles af databaseadministrator og/eller udvikler. Dette gøres ved hjælp af en række constraints, det vil sige forskellige former for begrænsning på de kolonneværdier, som det er muligt at indsætte.

Constraints har følgende fordele:

- Lette at skrive v.h.a. SQL kommandoer
- Reglerne for valide data ligger centralt i datadictionary, d.v.s alle data inddateret gennem applikationer vil overholde reglerne.
- Ændres en constraint, vil ændringen slå igennem på alle applikationer, der skal altså kun rettes et sted, og ikke i samtlige applikationer.
- Indtastes en ikke-valid værdi i et felt i en applikation, vil brugeren advares omgående, d.v.s. før SQL-sætningen udføres.
- Oracle optimizer bruger constraints og forbedrer dermed performance.
- Constraints kan gøres inaktive (disables), således at store datamængder kan indateres uden constraintcheck. Herefter kan constraints igen gøres aktive (enables), og alle data checkes. Alle data, der ikke overholder disse integritets-constraints, vises i en exceptions tabel.

Constraint-typer

Der findes følgende typer af constraints:

• NOT NULL	Angiver at kolonnen ikke må være tom.
• UNIQUE	Angiver at der ikke må være sammenfaldende værdier i kolonnen.
• PRIMARY KEY	Identificerer primærnøglen. Det indebærer,at der ikke må være sammenfaldende værdier i kolonnen, og der må ikke være Null-værdier.
• FOREIGN KEY REFERENCES	Identificerer fremmednøglen. Refererer til en tilsvarende kolonne i en anden tabel. Defineres på fremmednøgle-siden af to tabeller og vil fx. bevirke, at der ikke kan indsættes værdier, uden at en tilsvarende værdi eksisterer i primærtabellen. Omvendt vil det ikke være muligt at slette en række i primærtabellen, hvortil der er referencer i fremmedtabellen.
• CHECK	Angiver en betingelse, kolonnen skal opfylde for at blive indsat i tabellen, fx. at kolonneværdi ligger i et bestemt værdiområde.
• DEFAULT	ex hvis der indsættes NULL, indsat den default værdi er dags dato

Definition af constraints

Constraints defineres sammen med tabellen i en CREATE TABLE-kommando eller tilføjes med ALTER TABLE-kommandoen.

Constraints kan defineres på to måder. En constraint på en enkelt kolonne kan defineres på samme linie som definitionen af kolonnen, 'column constraint syntaks', eller på en anden linie 'table constraint syntaks'.

En constraint på flere kolonner kan kun defineres med table constraint syntaks.

Constraint på én kolonne

Eksempel

Opret tabellen dept, med en UNIQUE constraint på dname kolonnen.

Column constraint syntaks:

```
SQL>  CREATE TABLE dept (
              deptno NUMBER(2),
              dname VARCHAR2(12) CONSTRAINT uk_dept_dname UNIQUE,
              loc VARCHAR2(13)
              );
```

Table constraint syntaks:

```
SQL>  CREATE TABLE dept (
              deptno NUMBER(2),
              dname VARCHAR2(12),
              loc VARCHAR2(13),
              CONSTRAINT uk_dept_dname UNIQUE (dname)
              );
```

Constraint på flere kolonner

Eksempel

Opret tabellen dept, med en UNIQUE constraint på kolonnerne dname og loc.

Table constraint syntaks:

```
SQL>   CREATE TABLE dept (
              deptno NUMBER(2),
              dname VARCHAR2(12),
              loc VARCHAR2(13),
              CONSTRAINT uk_dept_dname_loc UNIQUE (dname,loc)
              );
```

Eksempler på typer af constraints

Eksempel 1

Tilføj defaultværdien 5000 til customer.creditlimit:

```
SQL>  ALTER TABLE customer
      MODIFY(creditlimit DEFAULT 5000);
```

Denne constraint sikrer, at kunder som udgangspunkt har en kreditgrænse på 5000.

Eksempel 2

Tilføj en validering af at customer.custid ligger i intervallet 100-800:

```
SQL>  ALTER TABLE customer
      ADD(CONSTRAINT ck_custid
      CHECK(custid BETWEEN 100 AND 800));
```

Denne constraint sikrer, at kundenumre ligger i intervallet 100-800.

Eksempel 3

Tilføj en primærnøgle til customer.custid:

```
SQL>  ALTER TABLE customer
      ADD(CONSTRAINT pk_customer_custid
      PRIMARY KEY(custid));
```

Denne constraint sikrer, at der er defineret et kundenummer for samtlige kunder, og at kundenumrene er unikke.

Eksempel 4

Tilføj en fremmednøgle på orders.custid der refererer til customer-tabellen:

```
SQL>  ALTER TABLE orders
      ADD(CONSTRAINT fk_orders_custid
      FOREIGN KEY(custid)
      REFERENCES customer(custid));
```

Denne constraint sikrer, at der ikke kan indtastes ordrer til en ikke eksisterende kunde, og at en kunde med ordrer ikke kan slettes.

Eksempel 5

Opret en afdelingstabel hvor følgende krav opfyldes:

- afdnr i intervallet 10-99
- adfnavn med store bogstaver
- placering i Herlev, Slagelse, Hobro eller Randers

```
SQL>  CREATE TABLE afdelinger (
              afdnr NUMBER
                    CONSTRAINT ck_afdnr
                    CHECK(afdnr BETWEEN 10 AND 99)
                    DISABLE,
              afdnavn VARCHAR2(20)
                    CONSTRAINT ck_afdnavn
                    CHECK (afdnavn=UPPER(afdnavn))
                    DISABLE,
              placering VARCHAR2(20)
                    CONSTRAINT ck_placering
                    CHECK(placering IN('HERLEV','SLAGELSE','HOBRO','RANDERS'))
                    DISABLE);
```

Eksempel 6

Constraint på emp-tabllen skal enables. Evt. rækker der ikke opfylder constraintets betingelser, skal skrives til en exceptiontabel kaldet ex_emp.

```
SQL>  ALTER TABLE emp
      ENABLE CONSTRAINT ck_sal
      EXCEPTIONS INTO ex_emp;
```

Med denne exception vil alle ikke valide rækker vises i en tabel, der på forhånd er oprettet med kolonnerne row_id, owner, table name og constraint. I dette tilfælde hedder tabellen ex_emp. Er der rækker der ikke opfylder betingelserne i de oprettede constraints, vil constraints forblive inaktive. Med exception-tabellen har man mulighed for at se, hvilke rækker det drejer sig om, og dermed mulighed for at ændre tabellen.

Eksempel 7

```
SQL>  ALTER TABLE emp
      MODIFY(empno NUMBER CONSTRAINT nn_emp_empno NOT NULL,
             hiredate DATE DEFAULT SYSDATE)
      ADD(CONSTRAINT fk_emp_deptno
             FOREIGN KEY (deptno) REFERENCES dept ON DELETE CASCADE)
```

FOREIGN KEY REFERENCES sikrer, at alle ansatte i emp tabellen arbejder i en afdeling, der findes i dept tabellen.

ON DELETE CASCADE har den effekt at samtlige medarbejdere i en afdeling slettes, hvis afdelingen slettes

Bemærk!

Ved ændring af tabeller med alter table er det kun en NOT NULL constraint, der kan oprettes med modify og 'column constraint syntaks'. Alle andre ændringer foretages med add og 'table constraint syntaks'.

Ved oprettelse af nye tabeller kan der kun anvendes 'table constraint syntaks' ved definition af en FOREIGN KEY constraint. For alle andre typer constraints, kan der frit vælges mellem 'table constraint syntaks' og 'column constraint syntaks'.

Constraints oprettes default som enabled. En constraint kan gøres aktiv eller passiv med en create table- eller alter table-kommando og en enable- eller disable-clause.

Data Dictionary Views

Oplysninger om constraints findes i følgende views:

- USER_CONSTRAINTS
- ALL_CONSTRAINTS
- DBA_CONSTRAINTS

Oplysninger om kolonner der indgår i constraints findes i følgende views

- USER_CONS_COLUMNS
- ALL_CONS_COLUMNS
- DBA_CONS_COLUMNS

eks problemer ved overgang fra sommertid til vintertid
Vil kunne håndteres i Oracle & ideal de kan gemmes i GMT men

update emp
set sal = 1001
where rowid in (select rowid from ex_emp);

Øvelse 6.1

ORACLE_SID = MTS
ora2/ora2

Formål: *at oprette og afprøve forskellige typer af constraints*

1. Opret en bruger og tildel den følgende roles: CONNECT og RESOURCE
2. Connect til brugeren og kør SQL-scriptet: /usr/ora7/forms30/demo/jltbld
3. Tilføj følgende defaultværdier til emp-tabllen:
   - dags dato til kolonnen emp.hiredate
   - 0 til kolonnen emp.comm

   alter table emp
   modify (hiredate default sysdate);

   Select * from user_constraints where table = emp;
   Select * from user_tab_columns —"— ;
4. Udfør følgende statement:

   INSERT INTO emp(empno,deptno)
   VALUES(7000,10);

   DATA_DEFAULT

   Spørg mod emp-tabellen.

   Oprettet med default
5. Opret en constraint på emp-tabellen, der checker at EMPNO ligger mellem 7000 og 7999. Indsæt to nye rækker med værdier for EMPNO på henholdsvis 8000 og 7999.

   alter table emp add constraint ck_empno check (empno between 7000 and 7999);
6. Udfør følgende statement:

   ```
   SELECT constraint_name,constraint_type,table_name,status,search_condition
   FROM   user_constraints
   WHERE  constraint_type = 'C'
   ```

   └ NOT NULL er en check-constraint
7. Drop den constraint der blev oprettet under punkt 5.

create table AFD (AFDNR number(2),
PLAATS varchar2(20)
constraint ck_plaats check (plaats in
('HARLEM', 'HAARLEM', 'HAARLEMMERMEER')) deferrable
);

create table ex_emp (ex_row rowid,
ex_owner
ex_table
ex_cons

Øvelse 6.2

Formål: *at oprette og afprøve forskellige typer af constraints*

1. Opret en primary key constraint på ORDID-kolonnen i ord-tabellen.
   Bemærk, at der automatisk oprettes et indeks på den/de kolonner, der oprettes en primary key constraint på.

   Hvad hedder indekset? Brug data dictionary viewet IND.

2. Opret en foreign key constraint på ORDID-kolonnen i item-tabellen med on delete cascade option.

3. Check oprettelsen af de to constraints i dictionary viewene USER_CONSTRAINTS og USER_CONS_COLUMNS.

4. Slet ordre nummer 621 i ord-tabellen, og check at ordrelinierne til ordren er slettet.

5. Opret en primary key constraint på DEPTNO-kolonnen i dept-tabellen.

6. Opret en foreign key constraint på DEPTNO-kolonnen i emp-tabellen uden on delete cascade option.

7. Slet afdeling 10 i dept-tabellen. Hvad sker? Hvorfor?

8. Slet afdeling 40 i dept-tabellen. Hvad sker? Hvorfor?

9. Opret en tabel 'afd' med kolonnerne AFDNR og PLACERING og en constraint der checker på at PLACERING indeholder en af værdierne HERLEV,VEJLE eller KLOKKERHOLM. Constrainten skal være passiv (disables).

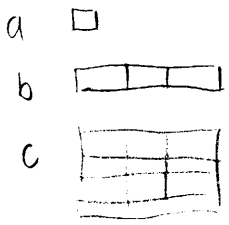
10. Indsæt værdier i afd-tabellen, således at HERLEV, VEJLE og KLOKKERHOLM er repræsenteret, men indsæt også et par rækker med andre bynavne.

11. Opret en tabel til exceptions med kolonnerne row_id, owner, table_name og constraints og gør check-constrainten på afd-tabellen aktiv (enabled). Husk at tage exception-delen med.

12. Hvad sker?
    Prøv at se i exceptions-tabellen.

    Prøv at joine afd-tabellen og exceptions-tabellen, for at se hvilke rækker der ikke overholder denne exception.

Declare




```
a   varchar(20) := 'HUGO';
b   emp%rowtype;
x   emp.ename%type;
cursor c is select * from emp order by
```

Hoe kennen wat 10% for de 5 laagst betaalde i emp

```
Update emp
set sal = sal * 1,10
where rowid in (select rowid from emp where rownum<6 order
                by sal);
```

MAN KAN IMMER BEREKEN ORDERBY i SUB-SELECT

```
Declare a number(8);
Begin   open c;
        for i in 1..6 loop
            fetch c into a;
                update emp set sal = sal*1,1 where empno= a;
        end loop;
        close c;
End;
```

7. Introduktion til PL/SQL

Programmeringssproget PL/SQL er en udvidelse til SQL sproget og giver en øget fleksibilitet i udførelsen af SQL sætninger. PL/SQL anvendes blandt andet til at skrive databasetriggere.

Pl/SQL giver mulighed for:

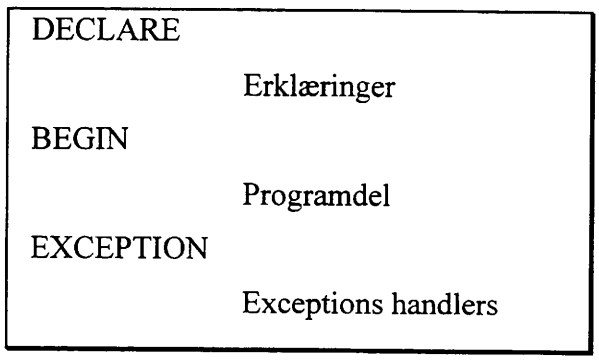
- betinget udførelse (IF..THEN..ELSE, EXIT, GOTO)
- repeteret udførelse (FOR..LOOP, WHILE..LOOP)
- oprettelse af variabler og konstanter
- tildeling af værdier

Desuden øges performance, idet flere SQL-sætninger kan sendes i én blok til oracle-kernen på en gang, i modsætning til afviklingen af SQL-sætninger uden PL/SQL, der kræver et kald til kernen for hver sætning. Forbedringen i performance er størst i et netværksmiljø.

PL/SQL blokken

Den grundlæggende programmeringsenhed i PL/SQL er en blok. Blokke kan nestes i en blok, det vil sige en eller flere blokke kan ligge inden i en anden blok. En blok med PL/SQL-kode inddeles med nøgleord i tre dele:

- Erklæringsdel
- Programdel
- Exception handler del



PL/SQL-blok

Erklæringsdelen

Her erklæres variabler og konstanter. Konstanter tildeles en fast værdi. Variabler kan tildeles en værdi her med tildelingsoperatoren :=. Variabler kan også tildeles en værdi i programdelen med en select-sætning.
Erklæringsdelen kan udelades af PL/SQL-blokken.

Eksempel

Erklær variablen navn som en tekst på 14 karakterer og variablen ny som tal og med værdien 0:

```
DECLARE
    navn VARCHAR2(14);
    ny NUMBER := 0;
              :
```

Hvis variablen navn skal have samme længde og datatype som en eksisterende kolonne :

```
DECLARE
    navn dept.dname%TYPE;
              :
```

Eksempel

Erklær konstanten moms til en værdi på 25%:

```
DECLARE
    skat CONSTANT NUMBER (3,2) := 0.25;
              :
```

Programdelen

Programdelen er den eneste del en PL/SQL-blok skal indeholde. Hvis fx en trigger kun består af en blok med kode, kan nøgleordene DECLARE og EXCEPTIONS udelades.

Eksempel

Erklær en variabel som skal bruges i programdelen:

```
DECLARE
    navn dept.dname%TYPE;
    CURSOR c1 IS SELECT dname FROM dept;
BEGIN
    OPEN c1;
        LOOP
            FETCH c1 INTO navn;
            EXIT WHEN c1%NOTFOUND;
            INSERT INTO........
                :
```

Exceptiondelen

Exceptions bruges til håndtering af unormale hændelser ved afvikling af en SQL-sætning. Exceptiondelen kan udelades fra PL/SQL-blokken.

Typiske exceptions er:

- at der ikke returneres nogen rækker i en forespørgsel
- at flere rækker matcher søgekriterierne

Når en exception opstår, afbrydes afviklingen af SQL sætningen, og der springes til exceptiondelen, hvis den findes. I exceptiondelen behandles exceptions ved hjælp af exception handlers, der angiver alternative måder for afvikling af PL/SQL-blokken. Hvis der ikke findes nogen exception handler, der svarer til den type exception, der opstod, vil der returneres en fejl.

En exception handler skal altså svare til den exception, der opstår, for at kontrollen overgår til exceptiondelen. Den består af navnet på en exception og de statements, der skal udføres.

Exception: Hvilken fejl/uhensigtmæssighed

Exception handler: Hvad skal der ske (navn på exception+aktion)

Predefinerede exceptions

I PL/SQL ligger der en række predefinerede exceptions, det vil sige systemet automatisk signalerer en fejl/uhensigtsmæssighed. Det er dog også muligt at definere egne exceptions.

- DUP_VAL_ON_INDEX
  Signaleres, når der ved insert eller update, vil blive oprettet to rækker med de samme værdier i kolonner med en 'unik indeks constraint'.
- INVALID_NUMBER
  Signaleres, når en karakterstreng ikke kan konverteres til number, fordi den ikke er valid som tal.
- NO_DATA_FOUND
  Signaleres, når en SELECT sætning ikke returnerer nogen data.
- TOO_MANY_ROWS
  Signaleres, når en SELECT sætning returnerer mere end en række.
- OTHERS
  Signalerer alle de exceptions, der ikke er nævnt i exceptionhandlerdelen af PL/SQL-blokken.

raise
Listen er ikke fuldstændig. Alle predefinerede exceptions kan findes i PL/SQL User's Guide and Reference side 6-25.

20001 til 20999

raise_application_error(20001, 'Ingen data i cursor');

raise_form_trigger_failure

Eksempel

```
DECLARE
    navn dept.dname%TYPE;
    CURSOR c1 IS SELECT dname FROM dept;
BEGIN
    OPEN c1;
        LOOP
            FETCH c1 INTO navn;
            EXIT WHEN c1%NOTFOUND;
            INSERT INTO........
                :
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        INSERT INTO.........
    WHEN OTHERS THEN
        INSERT INTO.........
END;
```

Eksempel

Udskriv medarbejdernr, navn og løn på de 5 højst betalte medarbejdere.

```
SET ECHO OFF
DECLARE
   CURSOR c1 IS
   SELECT empno,ename,sal FROM emp
   ORDER BY sal DESC;
   navn VARCHAR2(10);
   mnr  NUMBER(4);
   loen NUMBER(8,2);
BEGIN
   OPEN c1;
   LOOP
      FETCH c1 INTO mnr,navn,loen;
      EXIT WHEN (c1%ROWCOUNT>5) OR (c1%NOTFOUND);
      INSERT INTO temp VALUES(mnr,navn,loen);
      COMMIT;
   END LOOP;
   CLOSE c1;
END;
/
SET FEED OFF
SELECT * FROM temp ORDER BY loen DESC;
TRUNCATE TABLE temp;
SET FEED ON
SET ECHO ON
```

Udskriften kan fx se således ud:

```
SQL> @max
SQL> SET ECHO OFF

PL/SQL-procedure er fejlfrit udført.

   MNR  NAVN                      LOEN
------- -------------------- -------
   7839 KING                    5000
   7902 FORD                    3000
   7788 SCOTT                   3000
   7566 JONES                   2975
   7698 BLAKE                   2850
```

```
while a<100 loop
    a := a+1;
end loop
```

```
if  a<0  then
    ===;
elsif  a>0  then
else
end if;
```

```
goto label1

<<label1>>
```

User @rors

```
declare
    Cursor   c1   is
        select customer name, ord.ordid , ord.total
        from Customer, ord where customer.custid =
                                 ord.custid

        order by ord.total desc

    c1_result  c1%rowtype ;

begin
    open c1 ;
    loop
        fetch c1 into c1_result ;
        exit when (c1%rowcount>10) OR (c1%NOTFOUND);
        insert into tmp values (c1_result.name, c1.result.
                                ordid, c1.result.total );

    end loop ;
          commit;
    close c1 ;
end ;
/

Select * from tmp order by total desc
/* truncate table tmp; */
```

Øvelse 7.1

pl 7_1.sql

Formål: at skrive og afprøve PL/SQL-kode

1. Udskriv firmanavn (customer.name), ordrenummer (ord.ordid) og ordretotal (ord.total) for de 10 største ordrer.

   Den færdige udskrift kan fx se således ud:

```
SQL> @maks
SQL> SET ECHO OFF

PL/SQL-procedure er fejlfrit udført.
```

cust_name

FIRMA	ORDRENR	TOTAL
K + T SPORTS	617	46,370.00
VOLLYRITE	614	23,940.00
SHAPE UP	605	8,324.00
NORTH WOODS HEALTH AND FITNESS SUPPLY CENTER	613	6,400.00
EVERY MOUNTAIN	612	5,860.00
JOCKSPORTS	620	4,450.00
VOLLYRITE	618	3,510.50
EVERY MOUNTAIN	619	1,260.00
JUST TENNIS	616	764.00
JOCKSPORTS	621	730.00

create table tmp (cust_name varchar2 (~~15~~)
ordid number (6)
~~total~~ total number (10,2)
)

column TOTAL total 9999999999.99

8. Stored Procedures

Procedurer er samlinger af SQL- og PL/SQL-statements, der udføres som en samlet enhed. Procedurer har bla. fordele på følgende områder:

- **Sikkerhed**
  Det er muligt at begrænse de database oprerationer brugerne kan udføre ved kun at tillade dem adgang til data gennem procedurer og funktioner. Eksempelvis kan brugerne grantes rettighed til at udføre en procedure, der opdaterer en tabel, uden at de på anden måde kan søge i eller manipulere med tabellen.
- **Performance**
  Da procedurer anvender PL/SQL, og dermed kan sende teksten i en hel PL/SQL-blok afsted samlet frem for at kalde Oracle-kernen for hvert statement, vil den mængde information der skal sendes blive reduceret kraftigt. Det er især en fordel på netværksløsninger.

  Procedurer gemmes i kompileret form i databasen, hvorfor kompilering ikke er nødvendig ved udførelse.

  Procedurer repræsenteres i Shared Pool.
- **Memory allokering**
  Procedurer bruger som nævnt Shared Pool. Det betyder, at mange brugere kan anvende den samme kopi.
- **Produktivitet**
  Hvis applikationer designes til at bruge et sæt almene procedurer, undgås redundant kodning. Viser det sig, at data skal behandles efter andre regler, er det nok at rette de enkelte procedurer og ikke alle applikationer.
- **Integritet**
  Det er kun nødvendigt at teste en procedure én gang, for at konstatere om den virker. Når det er gjort, kan den bruges af et vilkårligt antal applikationer, hvori den så også vil virke. Som nævnt ovenfor, vil en eventuel ændring kunne klares ved at rette og kompilere den enkelte procedure.

Lagring af procedurer

Når en procedure oprettes udfører Oracle automatisk følgende:

- Kompilerer proceduren.
  Hvis en fejl opstår retuneres en fejlmeddelelse.
- Gemmer den kompilerede kode
  Oracle cacher den kompilerede kode i Shared Pool. Koden forbliver i cachen så længe LRU-algoritmen tillader det, også selvom den bruger der oprindeligt kaldte proceduren afbryder sin session.

- Gemmer proceduren i databasen
  På oprettelsestidspunktet gemmer Oracle procedurens navn, en repræsentation af den kompilerede kode (parse tree), pseudo code (P code) genereret på baggrund af den parsede kode og eventuelle fejlmeddelelser.

Alle dele af procedurerne gemmes i system tablespace. Nærmere bestemt i Data Dictionary.

Opret procedure

En procedure oprettes med CREATE PROCEDURE-kommandoen. Bemærk at DECLARE-delen ikke er en del af syntaksen.

Eksempel

Opret en procedure der opretter en ny kunde i customer-tabellen ved angivelse af kundenr, firmanavn og id på tilknyttet repræsentant.

```
SQL>  CREATE PROCEDURE nykunde(knr NUMBER,navn VARCHAR2,sid NUMBER)
      IS
      BEGIN
          INSERT INTO customer(custid,name,repid)
          VALUES(knr,navn,sid);
      END;
      /
```

Proceduren nykunde afvikles således:

```
SQL>  EXECUTE nykunde(500,'Ketcher-baren',3567);
```

Eksempel

Slet en kunde ved at angive kundenr. Hvis en ikke eksisterende kunde forsøges slettet, skal en forklarende fejlmeddelelse returneres.

```
SQL>  CREATE PROCEDURE slet_kunde(knr NUMBER)
      IS
      BEGIN
         DELETE FROM customer WHERE custid=knr;
         IF SQL%NOTFOUND THEN
            RAISE_APPLICATION_ERROR(-20001,'Kunden eksisterer ikke');
         END IF;
      END;
```

Proceduren slet_kunde afvikles således:

```
SQL>  EXECUTE slet_kunde(500);
```

Udførelse af procedurer

Når en procedure skal ekseveres gør Oracle følgende:

- Verificerer brugerens tilladelser.
  Det kontrolleres at brugeren har EXECUTE-privilegie. Bemærk, at det ikke er nødvendigt, at brugeren har adgang til de objekter der refereres til i proceduren.
- Verificerer procedurens validitet.
  I datakataloget kontrollerer Oracle at status på proceduren er VALID. En procedurer skifter eksempelvis status til INVALID, hvis et eller flere af de objekter der refereres til er slettet. Det samme vil ske hvis et system- eller objekt-privilegie fratages ejeren af proceduren. Naturligvis kun under forudsætning af at privilegiet har berøring med proceduren.
- Eksekverer proceduren.

Øvelse 8.1

Formål: At oprette og afprøve procedurer

1. Opret en procedure, som kan slette en medarbejder fra emp-tabellen ved angivelse af medarbejdernummer.

   Sørg for at gemme med SAVE undervejs.

2. Tilret proceduren, så der gives en fejlmeddelelse, hvis det angivne medarbejdernummer ikke eksisterer.

   Brugerdefinerede fejlkoder skal ligge i området -20000 til -20999.

3. Afprøv proceduren

4. Spørg mod USER_OBJECTS og USER_SOURCE
   - Hvornår blev proceduren oprettet?
   - Hvilken status har proceduren?
   - Hvordan er proceduren lavet?

Øvelse 8.2

Formål: At oprette og afprøve procedurer

1. Opret en procedure, der ved angivelse af afdelingsnummer og en procentsats, kan udføre følgende:
   - Forøge/formindske lønnen med den angivne procent (fx 12%, -10%)
   - Kun opdaterer den angivne afdeling
   - Returnerer en fejmeddelelse hvis afdelingen ikke eksisterer eller ikke har nogen ansatte. Fx 'Afdeling 45 har ingen ansatte'.
2. Afprøv proceduren med både positive og negative procentsatser og med forskellige afdelinger

9. Funktioner

Funktioner og procedurer er ens, bortset fra at en funktion returnerer en værdi, det gør en procedure ikke. En funktion kan kan fra Oracle V7.1 bruges direkte i SQL-statements.

Eksempel

Opret en funktion der validerer om et cpr-nummer opfylder modulus 11 prøven.

```
SQL>  CREATE OR REPLACE FUNCTION cpr_ok(cpr varchar2)
      RETURN INTEGER
      AS
      BEGIN
         IF LENGTH(cpr)!=11 THEN RETURN 1;
         END IF;
         IF SUBSTR(cpr,7,1)!='-' THEN RETURN 1;
         END IF;
         IF RTRIM(LTRIM(TRANSLATE(cpr,'0123456789','          '))))!='-' THEN
            RETURN 1;
         END IF;
         IF (TO_NUMBER(SUBSTR(cpr,1,1))*4+
            TO_NUMBER(SUBSTR(cpr,2,1))*3+
            TO_NUMBER(SUBSTR(cpr,3,1))*2+
            TO_NUMBER(SUBSTR(cpr,4,1))*7+
            TO_NUMBER(SUBSTR(cpr,5,1))*6+
            TO_NUMBER(SUBSTR(cpr,6,1))*5+
            TO_NUMBER(SUBSTR(cpr,8,1))*4+
            TO_NUMBER(SUBSTR(cpr,9,1))*3+
            TO_NUMBER(SUBSTR(cpr,10,1))*2+
            TO_NUMBER(SUBSTR(cpr,11,1))) MOD 11=0 THEN RETURN 0;
         ELSE RETURN 1;
         END IF;
      END;
      /
```

Udvælg de personer hvis cpr-nummer ikke er indtastet korrekt:

```
SQL>  SELECT * FROM personer
      WHERE cpr_ok(cpr)=1;
```

version >= '7.1.16'
for at funktioner kan benyttes direkte
i select-udtryk. Før kun muligt
fra PL/SQL

10. Database-triggere

En databasetrigger er en brugerdefineret PL/SQL-blok tilknyttet en tabel. En databasetrigger udføres (fyres), når den trigges ef et statement på tabellen.

Triggere bruges fx til at:

- forhindre ugyldige transaktioner
- sikre referentiel integritet, hvor dette ikke er sikret med constraints
- logge hændelser under givne forudsætninger
  fx indsætte en række i en anden tabel, hvis værdien i en kolonne er mindre end xxx
- øge sikkerheden
  fx at forhindre adgang til en tabel udenfor normal arbejdstid
- indsætte afledte (ændrede) værdier i en anden tabel før inserts eller updates
- udføre synkrone refreshes på snapshots.

Forudsætninger

Før der oprettes triggere på en database, bør følgende scripts afvikles som brugeren sys:

- $ORACLE_HOME/rdbms/admin/catproc.sql
- $ORACLE_HOME/rdbms/admin/dbmsstdx.sql

De brugere, der skal oprette triggere, skal endvidere være tildelt et af følgende systemprivilegier: CREATE TRIGGER eller CREATE ANY TRIGGER.

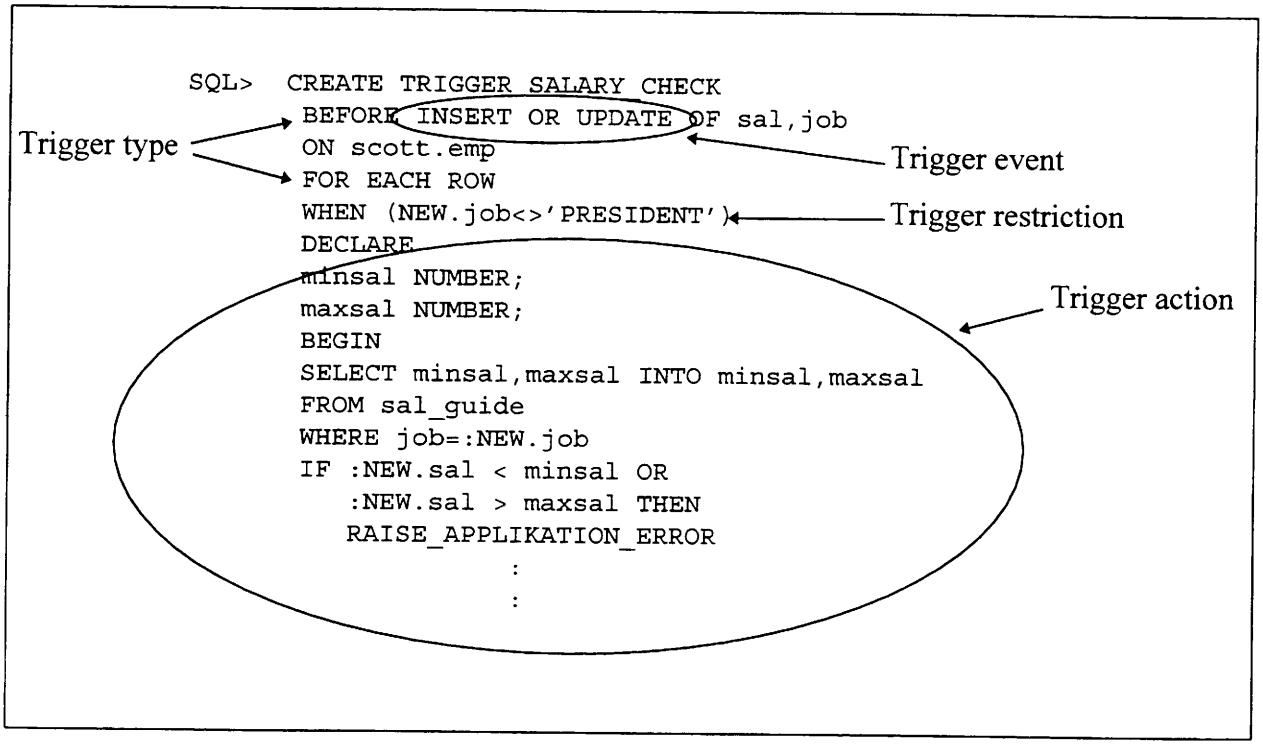
Bestanddele

En databasetrigger består af fire dele:

- Trigger type — BEFORE/AFTER, STATEMENT/ROW
- Triggering event — INSERT/UPDATE/DELETE
- Trigger restriction — WHEN-clause
- Trigger action — PL/SQL-blok

Se figuren på næste side.

Hver tabel kan have op til 12 databasetriggere - kombinationer af type og event. Det er altså ikke muligt fx at oprette 2 BEFORE/INSERT-triggere.



Egenskaber

Databasetriggere har følgende overordnede egenskaber:

- COMMIT, ROOLBACK og SAVEPOINT er ikke tilladt i databasetriggere.
- Brugere skal ikke have EXCUTE-privilegier på databasetriggere. En trigger vil altid fyre, når dens event og restriction er opfyldt, uanset hvilken bruger den trigges af.
- Databasetriggere er valide så længe tabellen eksisterer.
- Fyres uafhængigt af og i suplement til SQL*Forms triggere.
- Fyres af SQL-statements fra alle værktøjer og applikationer.

Opret databasetrigger

En databasetrigger oprettes med kommandoen CREATE TRIGGER.

Eksempel

Opret en trigger, der kun tillader ændringer i medarbejderoplyninger i arbejdstiden:

```
SQL> CREATE TRIGGER emp_permit_changes
     BEFORE
     DELETE OR INSERT OR UPDATE
     ON scott.emp
     DECLARE
       antal INTEGER;
     BEGIN
     /* Hvis det er lørdag eller søndag - returnér en fejl */
     IF (TO_CHAR(sysdate,'dy') IN ('SAT','SON') THEN
       RAISE_APPLICATION_ERROR(-20500,'Tabellen må ikke rettes i weekenden')
     END IF;
     /* Hvis dagen findes i tabellen over fridage - returnér en fejl */
     SELECT COUNT(*) INTO antal FROM firma_fridage
     WHERE dag=TRUNC(sysdate);
     IF antal >0 THEN
       RAISE_APPLICATION_ERROR(-20501,'Tabellen må ikke rettes på en fridag')
     END IF;
     /* Hvis tidspunktet er før 8 eller senere end 18 - returnér en fejl */
     IF TO_CHAR(sysdate,'HH24') not between 8 and 18 THEN
       RAISE_APPLICATION_ERROR(-20502,'Tabellen må kun rettes 8-18')
     END IF;
     END;
     /
```

Oracle vil fyre triggeren, hvergang et delete-, update- eller insert-statement vedrører scott's emp-tabel. Da triggeren er en before-trigger, vil Oracle oracle fyre triggeren inden det triggende statement.

Enable/disable triggere

En trigger har to distinkte tilstande:

- Enabled — En enabled trigger fyres, hvis type, event og restriction er opfyldt
- Disabled — En disabled trigger fyres slet ikke

Triggere disables midlertidigt i følgende situationer:

- Hvis et objekt, triggerne refererer til, ikke er tilgængeligt
- Hvis der skal indlæses store datamængder

En trigger er ved oprettelse enabled. Følgende kommandoer kan disable triggere tilhørende en tabel:

```
SQL>  ALTER TRIGGER <triggernavn> DISABLE;
SQL>  ALTER TABLE <tabelnavn> DISABLE ALL TRIGGERS;
```

Følgende kommandoer kan enable triggere tilhørende en tabel:

```
SQL>  ALTER TRIGGER <triggernavn> ENABLE;
SQL>  ALTER TABLE <tabelnavn> ENABLE ALL TRIGGERS;
```

OLD/NEW prefixes

Hver gang et statement indeholder en delete-, insert- eller update-kommando, kan man tale om, at en kolonne har en gammel og en ny værdi. Disse værdier kaldes af Oracle hhv. OLD og NEW. OLD og NEW kan kun bruges i row-triggere. Det er vedtaget at:

- OLD er NULL i inserts
- NEW er NULL i deletes

Eksempel

```
SQL>  CREATE TRIGGER SALARY_CHECK
      BEFORE INSERT OR UPDATE OF sal,job
      ON scott.emp
      FOR EACH ROW
      WHEN (NEW.job<>'RESIDENT')
      DECLARE
        minsal NUMBER;
        maxsal NUMBER;
      BEGIN
        SELECT minsal,maxsal INTO minsal,maxsal
        FROM sal_guide
        WHERE job=:NEW.job
        IF :NEW.sal < minsal OR
           :NEW.sal > maxsal THEN
           RAISE_APPLIKATION_ERROR(-20504,
           'Lønnen '||:NEW.sal||' for medarbejderen '||:NEW.ename||
           ' er uden for lønrammen for '||:NEW.job`);
        END IF;
      END;
      /
```

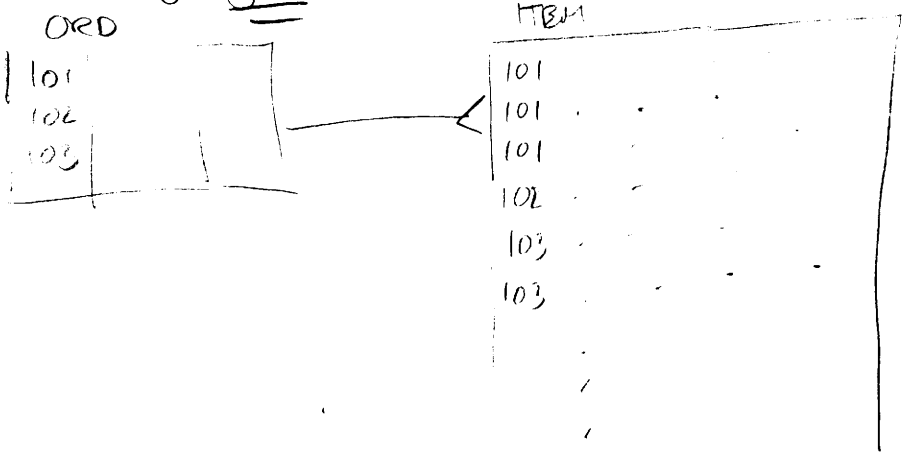
Øvelse 10.1

Formål: at oprette en databasetrigger
at bruge old/new-prefixes

1. Opret en databasetrigger på customer-tabellen. Triggeren skal sikre at en kundes kreditgrænse højst kan forøges med 10%

2. Triggeren skal returnere en fejlmeddelelse, der fortæller, hvis hvor stort et beløb der maksimalt må øges til, hvis grænsen er overskredet.

Trigger fyres af ~~for~~ constraints check'es




```
Update ord
Set ordid = ordid + 1000;
```

```
create trigger xxx
after update on ord
for each row
```

```
Update ITEM
    Set ORDID = :NEW.ORDID
    Where ORDID = :OLD.ORDID
```

Øvelse 10.2

Formål: at oprette en databasetrigger
at bruge old/new-prefixes

Overordnet opgavebeskrivelse:

Opret en databasetrigger, der for hver ændring af kolonnen emp.sal registrerer følgende oplysninger i en audit-tabel.

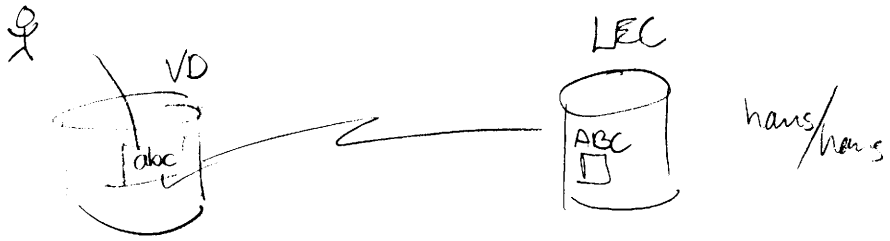
- Gammel løn NUMBER
- Ny løn NUMBER
- Medarbejdernavn VARCHAR2
- Dato/tidspunkt for ændringen VARCHAR2
- Oraclebruger, der udfører transaktionen VARCHAR2

1. Opret audit-tabellen
2. Opret databasetriggeren
3. Afprøv triggeren
4. Undersøg resultatet ved at spørge mod audit-tabellen
5. Spørg mod relevante data dictionary views

Database links :

Create database link Hugo
connect to HANS identified by HANS
USING 'T:109.178.92.101:S'
net host SID

Update GLR@label
set x = (select .... from abc@Hugo



Create public Synonym abc for abc@Hugo

11. Distribuerede databaser

En distribueret database er en samling af databaser, der for brugeren fremtræder som én database. Forbindelsen mellem de enkelte databaser muliggøres ved at bruge et netværk. Hver database kontrolleres af sit lokale DBMS (Database Management System). Alle databaseservere i den distribuerede database samarbejder for at opretholde konsistens i den globale database.

En databaseserver er den software, der administrerer en database. En client er en applikation, der søger information fra en server. Hver computer i et distribueret system kaldes en node. En node kan således være en client, en server eller begge dele.

Objekt-navne

For at kunne udvælge refererede objekter entydigt indenfor en enkelt database, har Oracle valgt en hierarkisk indfaldvinkel. Oracle garanterer, at alle brugere er unikke, og at alle objekter indenfor hver bruger er navngivet unikt. Hvis dette system udvides til, at databaser i et distribueret system er unikt navngivet, kan alle objekter i systemet refereres entydigt.

Gennemsigtighed

Med gennemsigtighed menes, at brugerne ikke ved og heller ikke behøver at vide, hvilken database et givent datasæt er placeret på. De værktøjer, der bruges til at opnå dette er views, synonymer og procedurer.

Inden disse værktøjer eksemplificeres, vil det nok være passende at se på, hvordan statements kan bruges til referere til objekter via SQL*Net V2.

Logon til databaseserver

Under forudsætning af, at databaseserveren er kendt af lytteprocessen tnslsnr (dvs. defineret i filerne listener.ora og tnsnames.ora beliggende i kataloget $ORACLE_HOME/network/admin), kan logon til en databaseserver foretages.

Eksempel

Kald SQL*Plus som brugeren ora på databasen MTS:

```
$>  sqlplus ora/ora@MTS
```

Bemærk, at en eventuel subshell vil blive startet på den lokale maskine.

Reference til tabel på databaseserver

For at kunne referere til en tabel på en databaseserver er det nødvendigt først at oprette et database link.

Eksempel

Opret et database link til Ora's words-tabel på databaseserveren MTS:

```
SQL>  CREATE PUBLIC DATABASE LINK wordmts
      CONNECT TO ora IDENTIFIED BY passwd
      USING 'MTS';
```

Ora's words-tabel på MTS-basen kan nu refereres således:

```
SQL>  SELECT * FROM words@wordmts;
```

Data dictionary views: USER_DB_LINKS, ALL_DB_LINKS og DBA_DB_LINKS.

Brug af views

Views giver mulighed for at sammenkøre oplysninger fra flere databaser, uden at dette er synligt for brugerne.

Eksempel

```
SQL>..CREATE VIEW firma AS
      SELECT empno,ename,sal,dname
      FROM   scott.emp e, jlt.dept@afd d
      WHERE  e.deptno=d.deptno;
```

Brugerne kan nu spørge mod tabellerne således:

```
SQL>  SELECT * FROM firma;
```

Brug af synonymer

Synonymer er i al sin enkelthed alternative navne til eksisterende objekter. Synonymer kan således bruges til at navngive en tabelreference, således at det bliver usynligt for brugeren, om tabellen ligger lokalt eller på en databaseserver.

Eksempel

Opret et synonym for scott's emp-tabel på MTS-basen:

```
SQL>  CREATE PUBLIC SYNONYM emp
      FOR scott.emp@MTS;
```

Der kan nu søges i tabellen således:

```
SQL>  SELECT * FROM emp;
```

Brug af procedurer

Der kan i procedurer refereres til objekter i eksterne databaser, uden at den bruger eller applikation, der afvikler proceduren, skal vide noger om det.

Eksempel

Opret en procedure der sletter en medarbejder i emp-tabellen på MTS-basen:

```
SQL>  CREATE PROCEDURE fire_emp(enum NUMBER)
      AS
      BEGIN
         DELETE FROM scott.emp@MTS
         WHERE empno=enum;
      END;
```

Hvis der er oprettet et synonym 'emp' for 'scott.emp@MTS' kan proceduren oprettes således:

```
SQL>  CREATE PROCEDURE fire_emp(enum NUMBER)
      AS
      BEGIN
         DELETE FROM emp
         WHERE empno=enum;
      END;
```

Hvis den refererede tabel omdøbes eller flyttes, er det kun det lokale synonym der skal rettes. Det er altså ikke nødvendigt at rette procedurer og applikationer, der kalder proceduren.

RECO-processen

Muligheden for en netværksfejl eller en hardwarefejl er altid til stede. Oracle garanterer, at hvis en sådan fejl opstår under commit af en distribueret transaktion, vil transaktionen automatisk blive håndteret globalt. Det betyder, at alle noder enten udfører et commit eller et rollback, når netværk eller system igen kommer op.

Baggrundsprocessen RECO genopretter automatisk verserende eller 'in-doubt' distribuerede transaktioner. Når en netværksfejl er rettet, bestemmer RECO-processerne for de involverede databaser udfaldet af enhver verserende transaktion

Distribuerede statements

Som eksempel vælges en distribueret forespørgsel. En distribueret forespørgsel bliver i den lokale database opdelt i en række enkeltforespørgsler. De enkelte forespørgsler bliver så sendt til de respektive noder og eksekveres der. Implicit afvikles et ALTER SESSION-statement, der sikrer at de sessions, der er forbundet, kører med samme karakteristik (fx nls_language). Efter eksekvering sendes resultaterne tilbage til den lokale node, som udfører den nødvendige efterbehandling. Dernæst returneres resultaterne til den kaldende bruger eller applikation.

Two-phase commit

Oracle kontrollerer og overvåger automatisk commits og rollbacks i en distribueret transaktion og opretholder integriteten i den globale database. Denne mekanisme kaldes 'two-phase commit'. Det eneste, der kræves, er at delprogrammet 'Oracle Distributed Option' er installeret.

Two-phase commit bruges kun, hvis en transaktion indeholder referencer til objekter på andre noder.

Alle operationer, som udføres af constraints, procedurer og triggers, er beskyttet af two-phase commit

Two-phase commit er opbygget af to adskilte faser:

- Forberedelsesfasen
- Commitfasen

Forberedelsesfasen

Alle refererede noder i en distribueret transaktion undtaget én (commit point site), bliver bedt om at forberede sig. At forberede sig vil sige, at hver node:

- allokerer de nødvendige resourcer til at udføre commit
- flusher redoinformation om alle transaktionens ændringer til den lokale redolog
- sender en af følgende beskeder til den refererende node: 'prepared', 'read-only', 'abort'

Den forberedte node venter nu på en commit- eller rollback-kommando. En forberedt node siges at være 'in-doubt'.

Hvis en node ikke kan gennemføre forberedelsesfasen udføres følgende:

- alle transaktionens ressourcer frigives, og der udføres rollback på den lokale del af transaktionen
- der svares tilbage med 'abort'
- et globalt rollback

Commitfasen

Hvis alle implicerede noder har gennemført forberedelsesfasen (returneret beskeden 'prepared'), fortsættes med commitfasen. I commitfasen sker følgende:

- alle noder bedes om at udføre commit
- Oracle udfører commit på hver node og lægger en oplysning i den lokale redolog om at, transaktionen er committet.

Pending Transaction Table

Alle Oracledatabaser har en 'Pending Transaction Table'. Tabellen gemmer information om distribuerede transaktioner, mens der udføres two-phase commit på dem. Tabellens indhold kan læses gennem data dictionary viewet DBA_2PC_PENDING.

Kolonner af særlig interresse er:

- **STATE**
  Transaktionens tilstand. Tilstanden beskrives med følgende nøgleord:
  1. Collecting
     Tilføjes kun til koordinatorer. Angiver, at noden samler data fra andre databaseservere, før den kan afgøre, om forberedelsesfasen kan gennemføres.
  2. Prepared
     Angiver, at noden er forberedt, men ikke om beskeden 'prepared' er afsendt. Ingen commit-kommando er modtaget.
  3. Committed
     Noden har committet transaktionen. Andre noder har muligvis ikke committet.
  4. Heuristic Commit
     Problemløsende commit. En verserende transaktion kan forceres til at committe efter databaseadministratorens skøn. Dette nøgleord bruges, hvis transaktionen lokalt committes manuelt.
  5. Heuristic Abort
     Problemløsende rollback. En verserende transaktion kan forceres til at udføre en rollback efter databaseadministratorens skøn. Dette nøgleord bruges, hvis transaktionen lokalt føres tilbage manuelt.

- **MIXED**
  Angiver, om de forskellige noder har samme udfald på transaktionen. Hvis det ikke er tilfældet, tildeles værdien 'YES'. Efter en fejlsituation kan databaseadministratoren eksempelvis skønne, at transaktionen skal føres tilbage, mens de andre noder har committet.

Snapshot

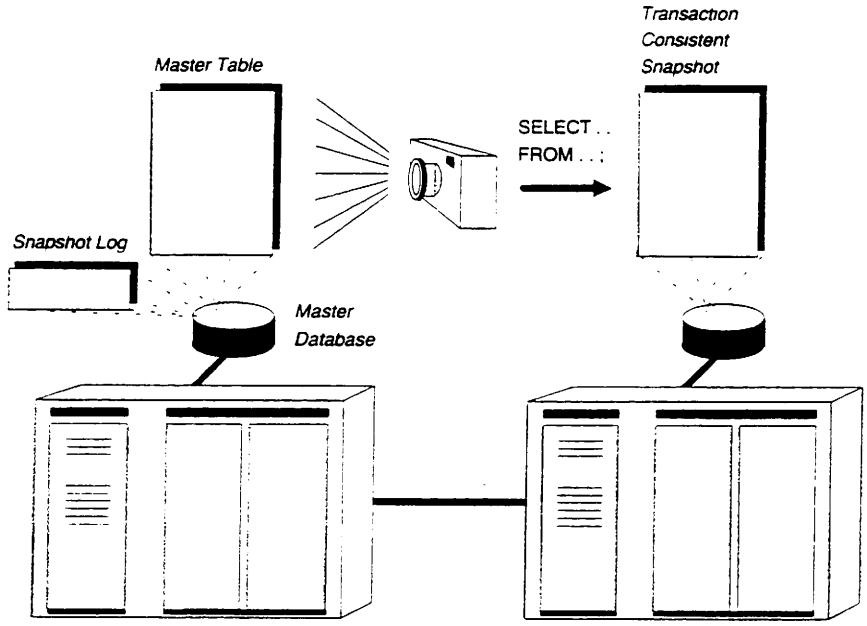
En distribueret arkitektur skal også kunne tilbyde faciliteter til transparent at replikere data mellem noderne i systemet. Den indbyggede snapshot-facilitet i Oracle tillader et ubegrænset antal tabelreplikationer af master-tabellen på andre noder i en distribueret database.

Opdateringer er kun tilladt på master-tabellen. Hver tabelreplikation kaldes et snapshot. Et snapshot er en fuld kopi af hele tabellen eller en delmængde af tabellen, der reflekterer en nylig udgave af master-tabellen..

Snapshot bruges, når der ofte spørges mod en tabel fra andre databaser. Især hvis tabellen sjældent opdateres.

Snapshots af en master-tabel bruges fordi:

- Forespørgsler kan udføres mod et lokalt snapshot. Det øger performance, da data ikke skal via netværket.
- Forespørgsler kan udføres trods netværksfejl



Updatering af et snapshot (refresh) kan udføres synkront, asynkront og manuelt.

Synkront refresh

Synkront refresh vil sige at snapshots opdateres når master-tabellen opdateres. Denne funktionalitet kan kun opnås ved brug af databastriggere. Denne funktionalitet kræver en del administration og vil ikke blive omtalt nærmere. Hvis emnet har interesse, kan konkrete eksempler på, hvordan det gøres, findes i ORACLE7 Server Application Developer's Guide side 12-11.

Asynkront refresh

Asynkront refresh vil sige, at snapshots opdateres automatisk med brugerdefinerede intervaller. Udføres af Oracle og er derfor en del af CREATE SNAPSHOT-kommandoen.

Eksempel

Opret et snapshot, der updateres hver mandag kl. 15.

```
SQL>  CREATE SNAPSHOT all_emps
      PCTFREE 5 PCTUSED 60
      TABLESPACE users
      STORAGE (INITIAL 50K NEXT 50K)
      REFRESH COMPLETE START WITH sysdate
                      NEXT NEXT_DAY(TRUNC(sysdate),'MONDAY')+15/24
      AS
      SELECT * FROM emp@jlt
      UNION
      SELECT * FROM emp@dhh;
```

For at kunne oprette snapshots skal brugeren have systemprivilegiet CREATE SNAPSHOT. Inden snapshots kan oprettes, skal de nødvendige database-links endvidere være oprettet. I eksemplet ovenfor forudsættes de to databaselinks jlt og dhh at være oprettet.

Asynkront refresh kræver desuden, at der findes en baggrundsproces til at udføre det. Baggrundsprocessen er navngiver SNPx. Baggrundprocesser, der varetager refresh af snapshots, startes ved at angive initialiseringsparameteren SNAPSHOT_REFRESH_PROCESSES til en værdi forskellig fra nul.

Manuelt refresh

Manuelt refresh udføres ved at udføre følgende kommando:

```
SQL>  EXECUTE dbms_snapshot.refresh('<snapshot>','C');
```

Ændring af snapshot

Snapshot kan ændres ved hjælp af ALTER SNAPSHOT-kommandoen. Dette gælder dog ikke det select-statement, der indgår i et snapshot. Ønskes det angivne select-statement rettet, skal snapshottet slettes og oprettes på ny. Et snapshot droppes kun på den base, hvor dropkommandoen gives.

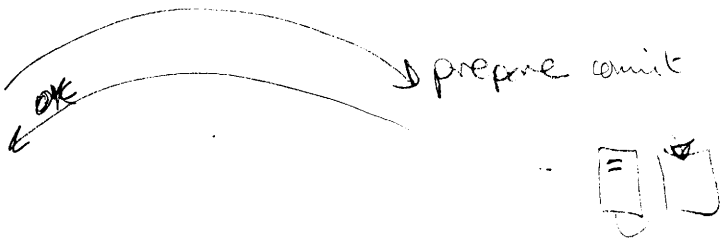
Eksempel

Snapshot 'qqq' skal ændres til at udføre refresh hver time:

```
SQL>  ALTER SNAPSHOT
      REFRESH COMPLETE NEXT ROUND(sysdate,'HH24');
```

flyt 40$ fra KBH-konto til NewYork

update A set x = x-40
update A@NY set x = x+40
commit;



DBA_2PC_NEIGHBORS
- IN- DOUBT

Øvelse 11.1

Formål: *at forespørge mod data på andre baser*
at oprette database links
at oprette et snapshot

1. Prøv at connecte dig til en bruger på en anden database.
2. Opret et database-link til en tabel på en anden base. Prøv dernæst at forespørge mod tabellen.
3. Opret et synonym, så forespørgslen under punkt 2. kan udføres uden reference til et database-link.
4. Opret et snapshot på en tabel fra en anden base, der har en refreshperiode på en uge.
5. Spørg mod det oprettede snapshot.
6. Spørg mod relevante data dictionary views.
7. Sørg for at master-tabellen opdateres.
8. Spørg igen mod det oprettede snapshot.
9. Udfør et manuelt refresh.
10. Spørg igen mod det oprettede snapshot.

12. Analyze

Analyze-kommandoen benyttes mest i forbindelse med optimering. I Oracle7 er der to forskellige principper at optimere efter:

- Cost-based optimering

  Bygger på objekt-statistik. Kræver derfor, at alle tabeller, indeks og clustre analyseres. Hvis størrelse af og datafordeling i objekter ændrer sig regelmæssigt, skal objekterne også analyseres regelmæssigt for at sikre, at statistikken repræsenterer det faktiske indhold.

- Rule-based optimering

  Bygger, som navnet angiver, på regler. Dette er Oracle's hidtidige optimeringsprincip.

Hvilket optimeringsprincip, der anvendes generelt på en database, bestemmes af initialiseringsparameteren OPTIMIZER_MODE. Denne parameter kan have følgende værdier:

- RULE Gennemtvinger rule-based optimering
- CHOOSE Vælger rule-based optimering, hvis statistik på de refererede objekter ikke findes. Ellers cost-based optimering

BEMÆRK:
Oracle anbefaler cost-based optimering og har derfor planer om at, rule-based optimering fjernes i kommende versioner. I Oracle V7.0 fungerer cost-based optimering dog ikke efter hensigten og kan derfor ikke anbefales i denne version.

Generering af statistik

Oracle genererer statistik ved brug af følgende teknikker:

- Estimate
  Estimat baseret på tilfældige dataprøver. Hurtig metode, der maksimalt læser 1064 rækker. Resultatet er oftest meget tæt på det eksakte resultat.
- Compute
  Beregner altid eksakte resultater. Tager længere tid end estimater. Tidsforbruger svarer til en 'full table scan' på tabellen.

Genereret statistik opsamles i data dictionary. Det drejer sig om følgende views:

- USER_INDEXES, ALL_INDEXES, DBA_INDEXES
- USER_TABLES, ALL_TABLES, DBA_TABLES
- USER_TAB_COLUMNS, ALL_TAB_COLUMNS, DBA_TAB_COLUMNS

På en tabel genereres bla. følgende statistik:

- Antal rækker
- Antal blokke i brug (compute)
- Antal blokke, der aldrig er brugt (compute)
- Gennemsnitlig tilgængelig friplads pr. blok
- Antal 'chained rows'
- Gennemsnitlige rækkelængde

Kolonne-statistik er antal distinkte værdier pr. kolonne.

Eksempel 1

Generér statistik på emp-tabellen med compute:

```
SQL>  ANALYZE TABLE emp
      COMPUTE STATISTICS;
```

Eksempel 2

Generér statistik på word-tabellen med estimate. Der skal estimeres på 2000 rækker:

```
SQL>  ANALYZE TABLE word
      ESTIMATE STATISTICS
      SAMPLE 2000 ROWS;
```

Eksempel 3

Generér statistik på word-tabellen med estimate. Der skal estimeres på 30% af tabellen:

```
SQL>  ANALYZE TABLE word
      ESTIMATE STATISTICS
      SAMPLE 30 PERCENT;
```

Angives 'sample' ikke, estimeres på maksimalt 1064 rækker. Angives 'sample' til mere end 50%, eller angives 'sample' til et antal rækker, der er mere end halvdelen, genereres statistikken i stedet med compute.

Validering af objekter

Validering kan foretages af tabel, index, cluster og snapshot. Valideringen verificerer integriteten af objektstrukturen. Der returneres kun en meddelelse, hvis der er registreret en fejl. Registreres en fejl, rettes den ved at droppe det pågældende objekt og derefter oprette det igen.

Eksempel 1

Udfør en validering af word-tabellen:

```
SQL>  ANALYZE TABLE word
      VALIDATE STRUCTURE;
```

Eksempel 2

Udfør en validering af word-tabellen og alle tilknyttede index:

```
SQL>  ANALYZE TABLE word
      VALIDATE STRUCTURE CASCADE;
```

Chained rows

Opsamling af statistik om 'chained rows' er lidt anderledes, idet den genererede statistik ikke placeres i data dictionary. Statistikken opsamles i en brugeroprettet tabel, der skal have et bestemt format. SQL-scriptet $ORACLE_HOME/rdbms/admin/utlchain.sql opretter en tabel ved navn 'chained_rows'.

Eksempel

Undersøg word-tabellen for 'chained rows':

```
SQL>  ANALYZE TABLE word
      LIST CHAINED ROWS
      INTO chained_rows
```

Øvelse 12.1

Formål: at bruge analyze-kommandoen til at generere statistik

1. Spørg mod USER_TABLES for at finde følgende oplysninger om emp-tabellen:
   - Antal tomme blokke
   - Gennemsnitlig friplads pr. blok
   - Gennemsnitlig rækkelængde
   - Antal rækker
2. Generer statistik på emp-tabellen med compute.
3. Gentag punkt 1.
4. Opret en tabel ved navn cr. Tabellen skal have følgende egenskaber:
   - Indeholde kolonnen navn VARCHAR2(20)
   - PCTFREE skal være 1
5. Indsæt kolonnen ename i tabellen cr, så mange gange at mindst 3 blokke er fyldt op, og udfør et commit.
6. Opdatér cr-tabellen så SMITH skifter navn til CHRISTOFFERSEN og udfør et commit
7. Undersøg om der er 'chained rows' i cr-tabellen.

www.cocom.dk

Image Scandinavia
Internet Service

---

explain plan for select * from emp

select * from plan_table

---

platinum technologies (Jima)
SQL-trace/sql debug (produkt)
Kundeusbaseret værktøj for debugging

Dansk Data Elektronik A/S
Herlev Hovedgade 199
DK 2730 Herlev
Tel.: (- 45) 42 84 50 11
Fax: (- 45) 42 84 52 20