Oracle Datenbank Architektur
- nicht nur für Einsteiger

Martin Klier
Senior Oracle Database Administrator
Klug GmbH integrierte Systeme, Teunz
martin.klier@klug-is.de

1 Einleitung

Eine Einführung in die Oracle-Datenbank-Basisarchitektur: Konzept, wichtige Begriffe, Verbindungsaufbau, Redo-Verfahren, Transaktionsabsicherung und Isolation. Zielgruppe sind primär Einsteiger in die Datenbanktechnik und IT-Fachkräfte, die nicht in Vollzeit als DBA arbeiten können.

2 Ziele

- „Das Konzept dahinter“ verstehen
- Primär für Einsteiger
- Vertiefung für Erfahrene

3 Inhalt

- Konzept: ACID
- Erstellen des Architekturschemas
    - Oracle Datenbank-Architektur
    - Verbindung, Namespace(s) und Abschottung
    - Instanz und Datenbank
    - Alles Blöcke, oder was?
    - Redo, Archivelogs, Undo und was man damit macht
- Oracle Architektur: Die Basis für neue Produkte: 10g,. 11g, 12c

4 Transaktionsprinzip „A.C.I.D.“

Hinter Transaktionen und Datenabrufen im Oracle Datenbankmanagementsystem steht das ACID-Prinzip: Die Abkürzung bedeutet „Atomicity, Consistency, Isolation and Durability“, zu Deutsch „Unteilbarkeit, Konsistenz, Abschottung und Dauerhaftigkeit“.

Eine Transaktion fasst in sich mehrere logisch voneinander abhängige, jedoch nicht zwangsweise technisch miteinander in Beziehung stehende Arbeitsschritte zusammen. Man könnte sie in der Alltagssprache eine geschlossene Handlung nennen. Das einfachste Beispiel ist die Überweisung von einem Bankkonto zum anderen, bei der keinesfalls die eine oder andere Seite Vorteile aus einem zwischen Abbuchung und Gutschrift vorkommenden Systemabsturz ziehen darf: Wenn die eine Aktion fehlschlägt, darf die andere nicht geschehen.

Gleichzeitig dürfen Leseoperationen durch andere Parteien auf dem selben Datenbestand keine in sich widersprüchlichen Resultate liefern: Die Daten müssen stimmig sein, obwohl sie konkurrierend gerade geändert werden.

Alle Transaktionen der Oracle Datenbank folgen dem ACID-Prinzip:

- **Atomicity** - ALLE Änderungen innerhalb der Transaktion werden ENTWEDER durchgeführt ODER verworfen.
- **Consistency** - die Datenbank wird von einem gültigen Status zu einem anderen überführt. Ein undefinierter Zwischenstand wird prinzipbedingt verhindert.
- **Isolation** - das Ergebnis der aktuellen Transaktion ist erst nach ihrem Abschluss für andere Operationen sichtbar.
- **Durability** - sobald die Transaktion geschlossen wurde, sind alle ihre Änderungen permanent verankert.

Das Wichtigste nochmals in einem Satz zusammengefasst: In einer Transaktion gilt alles oder nichts. Änderungen werden erst nach commit für andere zugänglich und (erst) was commitet wurde, ist sicher persistiert.

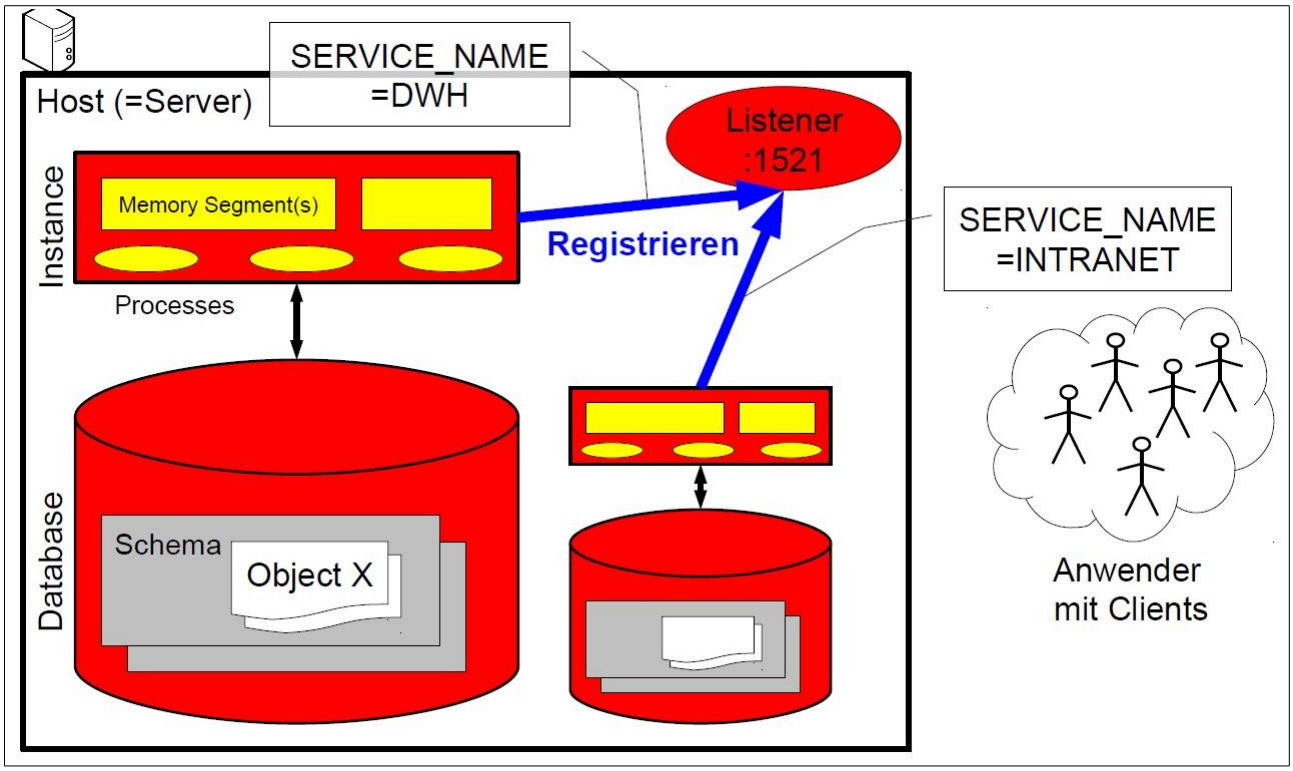
Darüber hinaus soll das System mit wenig Ressourcenaufwand die geforderten Schreib- und Leseoperationen möglichst schnell ausführen (=Performance). Die Oracle Database versucht, diese sich teils widersprechenden Anforderungen mit Hilfe einiger wichtiger Weichenstellungen im Kern zu lösen:

- Vermeidung von Sperren per Konzept - wo dennoch unumgänglich:
- Einsatz von möglichst schnellen Sperrmechanismen (Mutexes, die über native OS- oder ggf. auch Hardwarefunktionen aufgelöst werden können)
- Dezentralisierung (d.h. Kein zentraler Lock Manager), feingranulare Streuung der Serialisierungsstrukturen zur Vermeidung von Hot Spots
- Im Leerlauf unkomplizierte Abläufe, per Maschinenleistung skalierbare Mehraufwände bei höherer Komplexität
- Lasterzeugung nach dem Verursacherprinzip dort, woher die Belastung kommt.

5 Architektur

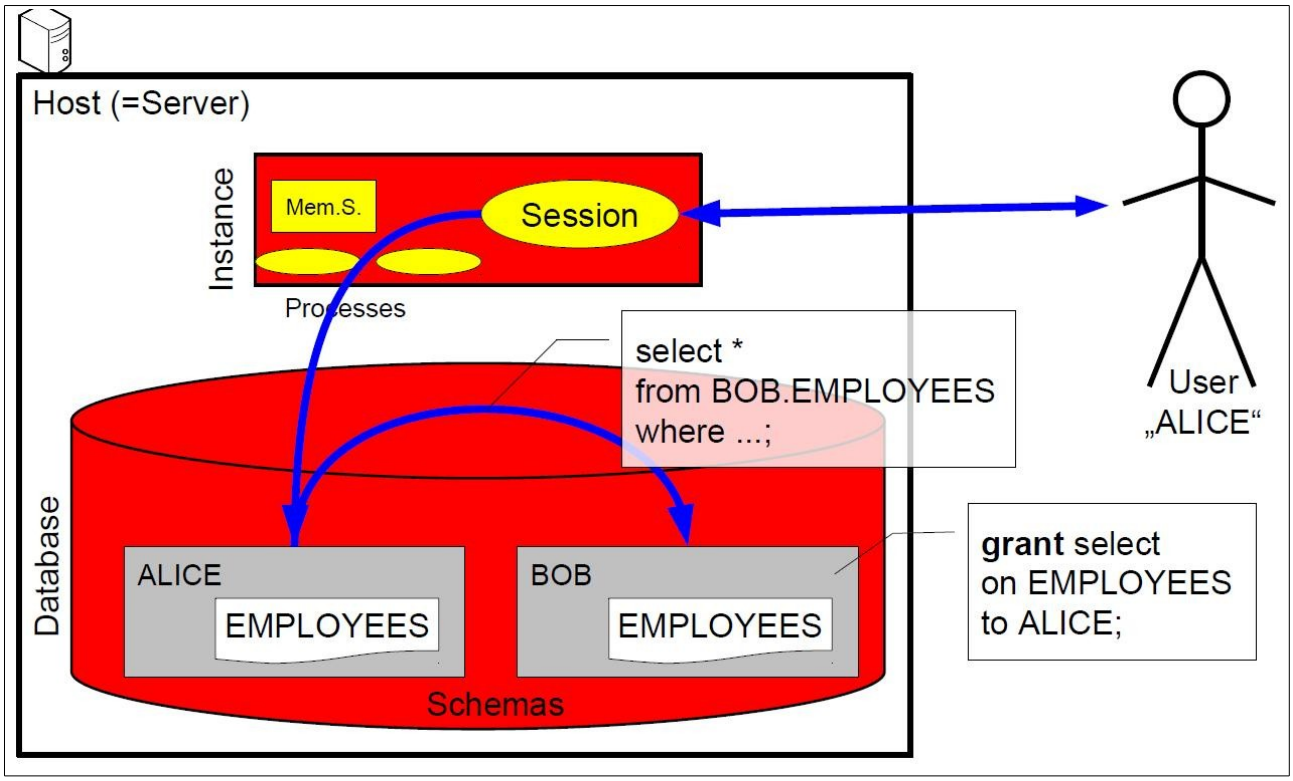
5.1 Grundbegriffe

Besonders im Vergleich zwischen Datenbanksystemen werden immer wieder Begriffe verwechselt bzw. mit unterschiedlicher Bedeutung unterlegt.



5.1.1 User, Schema und Abschottung

Der „User“ ist das Datenbankobjekt zur Unterscheidung von Personen oder mindestens Applikationen als Instrument der Sicherheit (Zugriffsschutz und Ausführungsberechtigungen). Authentifizierung über diverse Methoden; 1:1-Zuordnung zu einem Schema. Das „Schema“ ist ein Namespace für eine Menge von Datenbanksegmenten und anderen Objekten, die einem User zugeordnet werden. Der User ist automatisch Eigentümer der Objekte in seinem Schema.



Der Zugriff von einem Schema in ein anderes ist zunächst einmal nicht möglich, allerdings kann der Besitzer eines Objektes mit Hilfe eines „GRANT“ eine Zugriffsberechtigung aussprechen.

Codebeispiel:

```
BOB> grant select on EMPLOYEES to ALICE;
ALICE> select * from BOB.EMPLOYEES where NAME='SCOTT';
```

5.1.2 Datenbank

Die Summe aller dauerhaft gespeicherter Informationen auf dem Datenträger (z.B. Tablespaces, Redo Logs, ...), über Schemagrenzen hinweg.

5.1.3 Instanz

Die im RAM laufenden Prozesse und der ihnen zugeordnete exklusive und gemeinsam genutzte Speicher (z.B. PMON, SGA, Dedicated Server...)

5.1.4 Host

Das Computersystem/Server, auf dem das Datenbanksystem läuft und von dem aus es Benutzeranfragen beantwortet.

5.1.5 Listener

Ein Oracle-Prozess außerhalb der eigentlichen Instanz, der für das Annehmen von aufgebauten Verbindungen und dessen Weiterleitung an geeignete Workerprozesse/ -threads zuständig ist. Der Listener startet diese auch im Bedarfsfall. Sein TCP-Port ist üblicherweise :1521.

Die Instanz muss sich zunächst mit einem SERVICE_NAME am Listener registrieren, um überhaupt bedient zu werden. Dies geschieht automatisch, und wird über den Parameter LOCAL_LISTENER konfiguriert.

5.1.6 Datenbank-Link

Ein Datenbanksystem kann faktisch „als Client“ auf ein Zweites zugreifen, und verteilte Operationen ausführen. Objekte im entfernten System sind damit wie lokale nutzbar, z.B. für Änderungen, Lesevorgänge, Joins etc. Es gelten jedoch einige Einschränkungen für Datentypen und Skalierbarkeit, die den Rahmen dieses Dokumentes überschreiten, aber in der Oracle Produktdokumentation der jeweiligen Datenbankversion ausführlich dargestellt sind.

5.2 Interne Vorgänge

5.2.1 Blöcke, Buffer Cache und Lesevorgänge

Das Massenspeichermedium ist in Blöcke mit der Regelgröße 8k unterteilt. Sie enthalten zeilenbasierte Daten („Rows“) aus genau einem Segment (Tabelle, Index etc.). Diese Blöcke werden im Bedarfsfall 1:1 als „Buffer“ in den Buffer Cache im RAM eingelesen. Dies ist Aufgabe des dedizierten Serverprozesses der die anfordernde Session bedient. Einmal im Buffer Cache enthalten, ist der Buffer für alle anderen Sessions der Instanz ebenfalls ohne weitere Disk-IO nutzbar. Lesevorgänge auf Datensegmente erfolgen i.d.R. immer über Cache, um zu viele langsame IO-Operationen auf Disk zu vermeiden.

Da eine Datenbank im Normalfall wesentlich mehr Blöcke auf dem Massenspeicher ablegt als Buffer im Cache Platz finden, bestimmt das Datenbanksystem über ein spezielles Verfahren, welche Buffer am wenigsten benutzt wurden, und altert sie damit auch hinaus (Least Recently Used - LRU Algorithmus).

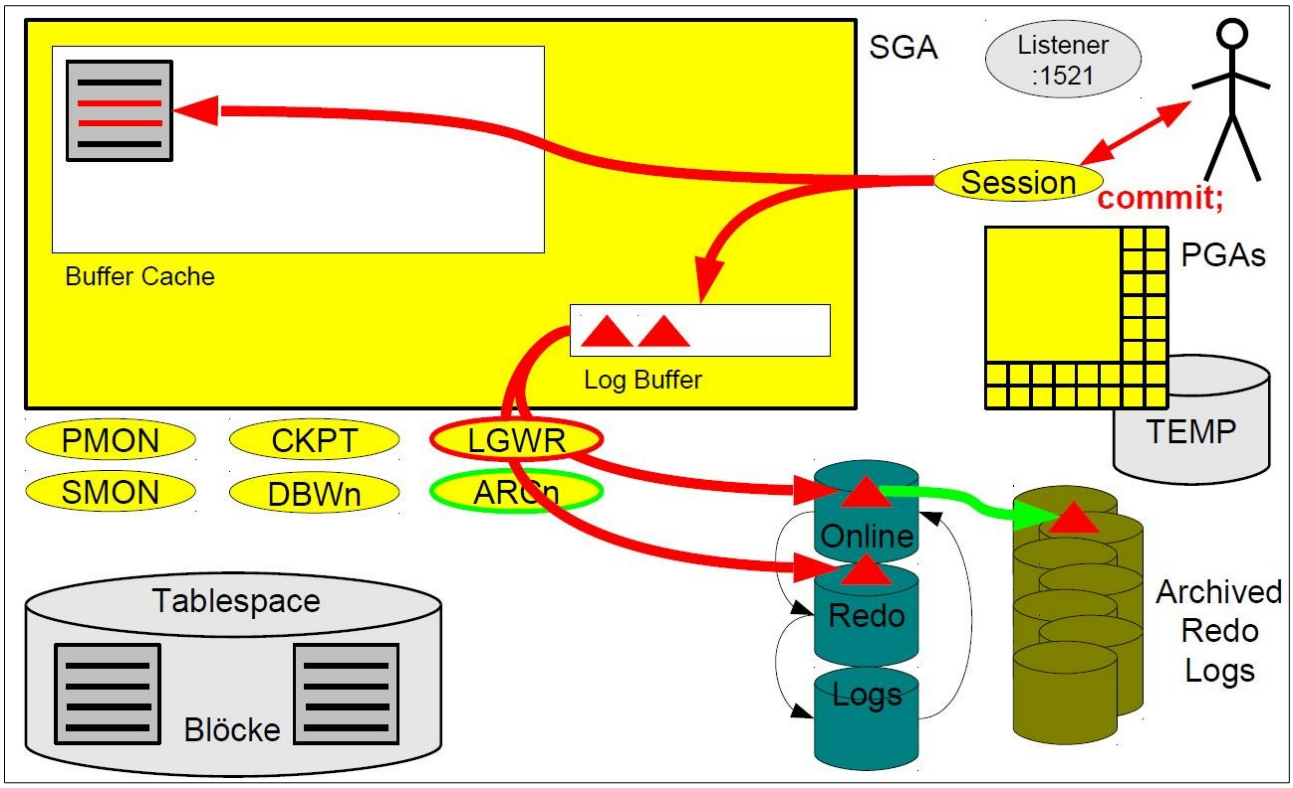
5.2.2 Daten ändern (Database Writer)

Auch die Änderung von Daten (UPDATE oder INSERT) erfolgt zur Vermeidung langsamer IO-Operationen i.d.R. nur innerhalb des Buffer Cache. Bevor der im letzten Absatz kurz beschriebene LRU-Algorithmus nun allerdings einen manipulierten Buffer aus dem Cache altern kann, muß dieser aber zunächst auf die Massenspeicher synchronisiert sein. Ansonsten ginge die Änderung verloren. Für das Persistieren geänderter Buffer auf Disk ist der Database Writer Prozess (DBWn) zuständig. DBWn ist aus Performancegründen ein „lazy writer“, der stets versucht IOs zu bündeln, und in Ruhezeiten des Systems optimiert zusammenhängende Blöcke zu schreiben. Dadurch ergeben sich allerdings ggf. erhebliche Verzögerungen und Differenzen zwischen Cache- und Disk-Inhalt. Es bleiben „dirty buffers“ stehen, die manipuliert und ggf. sogar commitet, aber nicht synchronisiert wurden.

5.2.3 Redo-Verfahren (Änderung und Crash Recovery)

Um trotz des im letzten Absatz beschriebenen Defizits des Database Writers, den Cache nur verzögert auf die Plattensysteme zu synchronisieren, die „Durability - Dauerhaftigkeit“ aus dem ACID-Prinzip zu gewährleisten, wurde das Redo-Verfahren mit Online Redo Logs entwickelt. Es sorgt dafür, dass alle Änderungen an Buffers möglichst schnell auf einen Massenspeicher gelangen.

Um aber nicht in die IO-Falle zu tappen die der DBWn durch das lazy writing zu vermeiden sucht, schreibt der zuständige Logwriter-Prozess (LGWR) nur Differenzen und keine vollen Blöcke in die Online Redo Logs. Diese schon erheblich reduzierte Datenmenge wird darüber hinaus nicht strukturiert sondern strikt sequentiell abgelegt, was sequentielles, schnelleres IO ermöglicht.



5.2.3.1 Ablauf

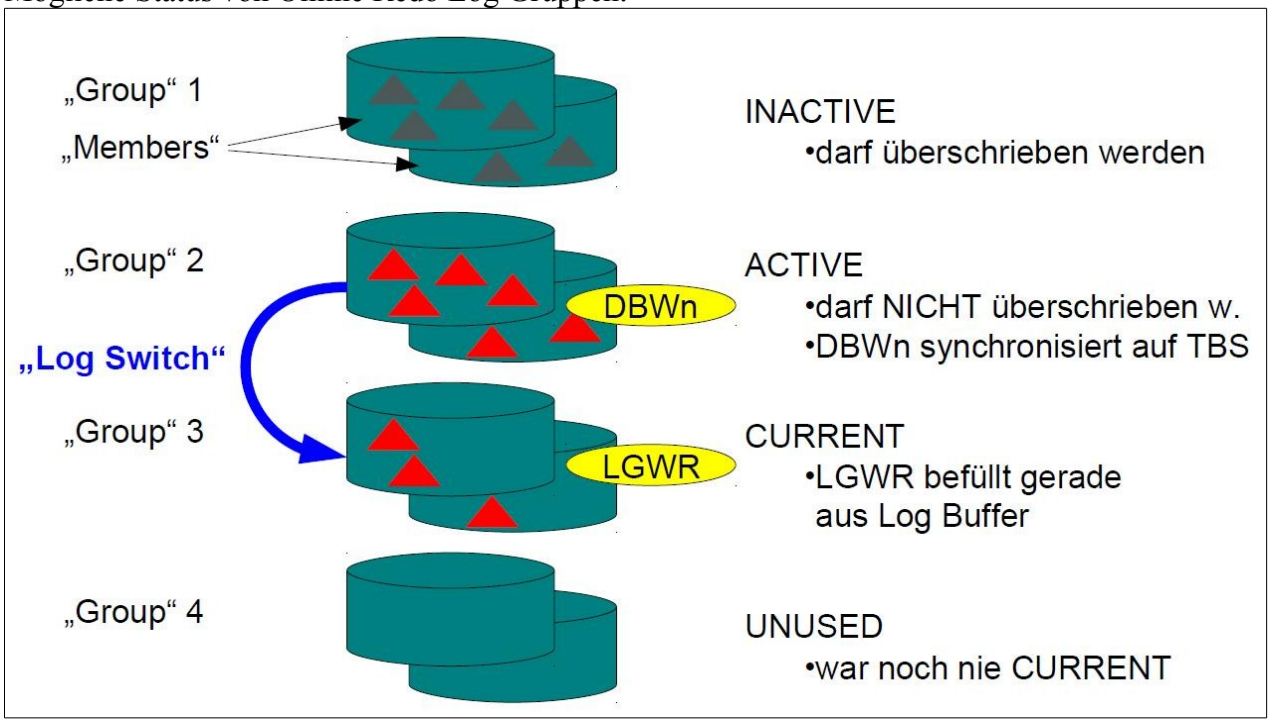
Der Ablauf ist recht einfach: Eine manipulierende Operation in einer Session muß nicht nur den betreffenden Block ändern, sondern auch die Differenz zwischen Vorher und Nachher in einen kleinen Cache schreiben, den Log Buffer. Ist der Log Buffer voll, sind drei Sekunden seit dem letzten Flush um oder erfolgt ein commit, so schreibt der LGWR den Inhalt in das aktuelle („CURRENT“) Redo Log.

Durch die Bindung des LGWR an das Commit-Event erfüllt der Weg über den Buffer Cache die Anforderung „Durability“ trotz des „lazy writers“ DBWn. Der Gewinn liegt in reduzierter IO-Menge im Vergleich zu einem synchron schreibenden Writers, insbesondere wenn sehr oft dieselben Buffers geändert werden.

5.2.3.2 Online Redo Logs

- dienen der Absicherung des Cache
- müssen aufbewahrt werden so lange DBWn nicht in Tablespace geschrieben hat
- müssen aufbewahrt werden so lange ARCn nicht in Archived Redo Log geschrieben hat (siehe Einschub; nächstes Unterkapitel)
- sind in „Groups“ gegliedert und mit „Members“ gespiegelt
- werden später zyklisch Überschrieben

Mögliche Status von Online Redo Log Gruppen:



5.2.3.3 Log Writer

- Schreibt den Log Buffer ins Online Redo Log
  - bei jedem commit
  - wenn Log Buffer voll
  - nach spätestens 3 Sekunden
- Laufzeit bestimmt Commit-Zeit
- Läuft mit höchster Priorität

5.2.3.4 Crash Recovery = Instance Recovery

Stürzt die Instanz ab, und konnte zuvor nicht alle „dirty buffers“ zurück auf die Plattensysteme schreiben, so enthalten die Online Redo Logs zwar alle commiteten Änderungen, diese sind aber für Userprozesse so nicht nutzbar. Die Instanz stellt jedoch beim Start fest, dass es Diskrepanzen zwischen den Datafiles (die die Blöcke enthalten) und dem letzten Stand der Redo Logs gibt. Die DB lädt dann die betroffenen Blöcke in den Buffer Cache und rollt sie mit Hilfe dazu eingelesener Redo-Sätze so weit vorwärts wie möglich. Erst wenn dieser „Rollforward“ abgeschlossen ist, wird die Datenbank geöffnet.

5.2.4 Einschub: Archivierung von Redo Logs

5.2.4.1 Verfahren

Archivierte Redo Logs haben zunächst nichts mit dem Absicherungsverfahren des DBMS zu tun, sondern gehören zum Bereich Backup und Recovery, den dieses Dokument nicht abdeckt. Dennoch kurz erklärt, stellen sie simple Kopien von Online Redo Logs dar, die vor deren Überschreiben durch den Archiver-Prozess (ARCn) angefertigt wurden. Mit Hilfe dieser Archive kann der Administrator die Lücke zwischen dem letzten Backup und der Gegenwart schließen, und damit ein „vollständiges Recovery“ durchführen. Grundsätzlich folgt dieses Verfahren aber genau dem Vorgang wie bereits beim Instance Recovery beschrieben: Blöcke mit „zu altem“ Stand werden mit Hilfe von Archived- und Online Redo Logs vorwärts gerollt.

5.2.4.2 Archived Redo Logs

- werden pro RL-“Group“ vom Archiver (ARCn) erzeugt
  - frühestens: nach Log Switch
  - spätestens: vor Überschreiben des jew. Online Redo Log
- dienen dem Nachfahren von verlorenen Operationen beim Recovery aus einem Backup
  - Complete Recovery
  - Point-in-Time-Recovery
- werden min. bis zum nächsten Backup aufbewahrt
  - nie überschrieben
  - später kontrolliert gelöscht

5.2.5 Undo

Das Undo-Verfahren schließt weitere Lücken im ACID-Prinzip. Es stellt die Mechanismen des Datenhandlings für Atomicity, Consistency und Isolation bereit, obgleich der eigentliche Code dafür natürlich in verschiedenen Operationen und Prozessen implementiert ist.

5.2.5.1 Konzept (rollback etc.)

Die schreibende Operation legt vorab in speziellen Buffers Einträge über die getätigten Veränderungen an, die sogenannten Undo-Records. Anders als die zuvor beschriebenen Redo-Informationen bezieht sich ein Undo-Record nicht auf den gesamten Block, sondern stets auf einen einzelnen Datensatz, eine Row.

„Undo“ bedeutet zunächst einmal nicht anderes als „rückgängig machen“. Das gleichnamige Verfahren ist vorrangig die Technologie die es uns erlaubt, angefangene Transaktionen mit dem Kommando „rollback“ rückgängig zu machen. Dazu rekonstruiert unsere Session mit Hilfe der zuvor von ihr angelegten Undo-Records den vorherigen Stand der veränderten Datensätze.

Den gleichen Mechanismus wendet die Oracle Datenbank auch an, falls die Session eines Users unerwartet beendet wird: Bei Verbindungsabbrüchen oder Abstürzen des zugehörigen Prozesses. Denn in diesem Falle kann unsere Session die Aufräumarbeiten naturgemäß nicht mehr selbst erledigen, dies geschieht hier stattdessen u.a. mit Hilfe des Hintergrundprozesses SMON.

Diese Funktionalitäten erlauben der Oracle Database, „Atomicity“ und „Consistency“ umzusetzen: Ohne zusätzliche Vorkehrungen zu treffen, kann der User sich darauf verlassen dass eine unabsichtlich unterbrochene Transaktion nicht in Teilen erhalten bleibt, sondern vollständig zurückgerollt wird. Optional kann er den Vorgang auch manuell auslösen.

Den Undo-Mechanismus nutzt die Datenbank, aus ganz ähnlichen Gründen wie ein User, für ihre internen Vorgänge ebenfalls: Damit ist „Consistency“ auch für die Metadaten gewährleistet.

5.2.5.2 Undo-Records

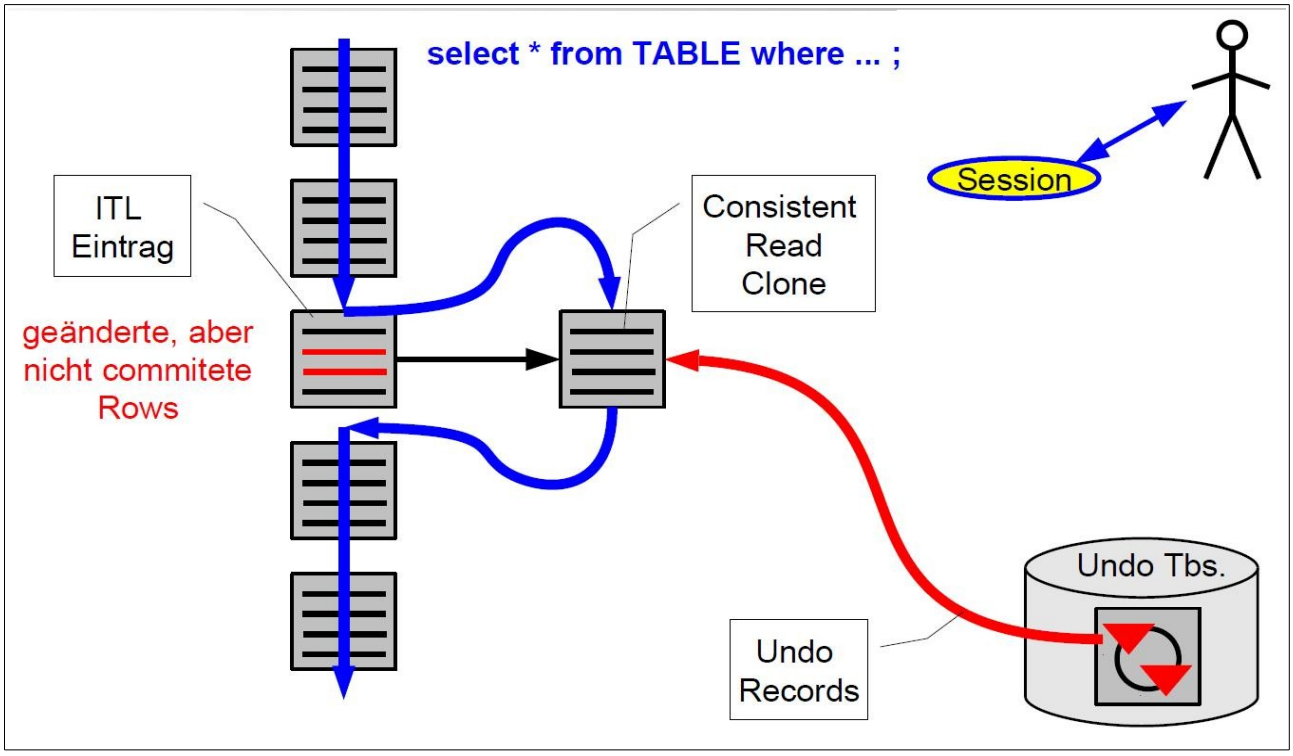
- dienen dem Zurückrollen von Änderungen
- werden in Buffers angelegt. Der zugehörige „Undo-Tablespace“ ist ein spezieller Typ von Tablespace
- kann die DB später überschreiben („Ringpuffer“)
    - frühestens: nach Abschluss der zugehörigen Transaktion
    - meist: nach Ablauf der Undo Retention
    - spätestens: bei Platzbedarf
- Es exisitiert kein „Undo-“Hintergrundprozess (d.h. Session / Job / etc. schreibt selbst)
- Weitere Verwendung
    - für konsistentes Lesen (stets)
    - Flashback-Technologien (optional)

5.2.5.3 Konsistentes Lesen

ACID bürdet dem Datenbanksystem eine weitere Last auf: „Isolation“ ist eines der am schwierigsten zu lösenden Probleme. Isolation bedeutet, dass veränderte Zeilen vor dem commit nicht für andere Sessions sichtbar werden dürfen. Denn dann würden deren Abfragen in sich nicht konsistente Daten anzeigen.

Im einfachsten Fall könnte man das Thema mit zwei Sperren lösen: Läuft gerade eine Abfrage, dürfte man diese Daten nicht ändern. Läuft gerade eine Änderung, dürfte man diese Daten nicht lesen. Für simple Anwendungen ohne konkurrierende Operationen würde dieses Modell ausreichen, obgleich das regelmäßige Prüfen einer Sperre („Polling“) sehr viel CPU-Zeit unproduktiv verschlingt.

Im Enterprise-Umfeld sind jedoch nur die wenigsten Fälle so flexibel, dass sie eine derartige 100%-Serialisierung tolerieren. Es ist in der Regel das Gegenteil gefragt: Man möchte völlig unabhängig voneinander Daten bearbeiten und ändern, und das ohne Wartezeiten und möglichst ohne dafür Ressourcen aufzuwenden. Ganz so einfach ist es zwar nicht, das Konzept von Oracle kommt dem aber schon nahe.



1. Ein Update legt neben den eigentlichen Zeilenänderungen die logischen Änderungen nochmals auch als Undo-Records an.
2. Möchte nun ein Select diese Rows lesen, erkennt er über die sogenannte „Interested Transaction List“, kurz ITL genannt, dass es eine nicht commitete Transaktion gibt, die diese Daten manipuliert hat.
3. ITL enthält auch die Information, wo die zugehörigen Undo-Records zu finden sind.
4. Nun kopiert der Lesevorgang den manipulierten Block im Buffer Cache, und rollt in der Kopie die veränderten Zeilen mit Hilfe der Undo-Records auf den Zustand vor der offenen Transaktion zurück. Damit entsteht der „Consistent Read Clone“.
5. Der Select liest die zurückgerollten Daten. Der Consistent Read Clone bleibt zur eventuellen weiteren Nutzung im Buffer Cache, bis er gemäß LRU verdrängt wird.

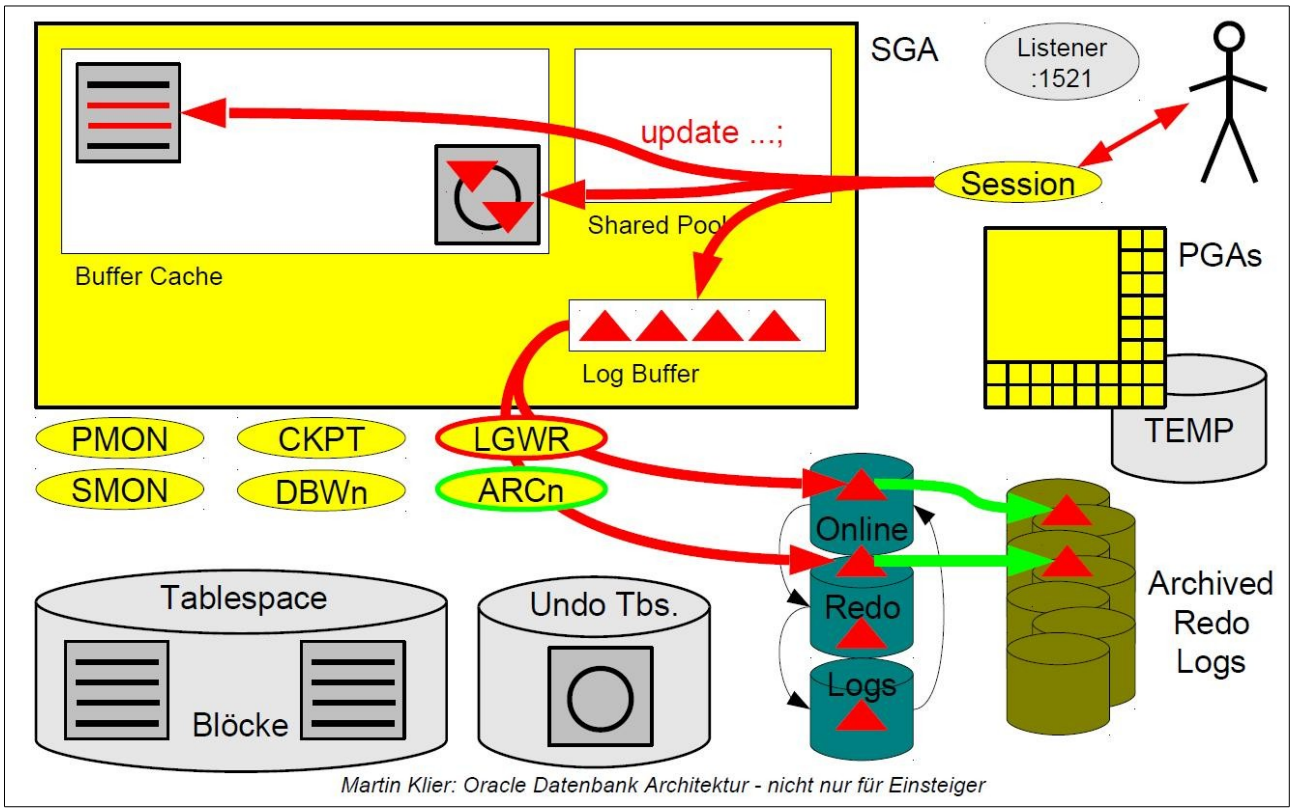
Damit kommt das Datenbanksystem ohne Lesesperren aus. Allerdings verursacht die Erzeugung des Consistent Read Clones einen gewissen Overhead bei dem Vorgang, der den konsistenten Lesevorgang anfordert. Der eigentliche Schreibvorgang hat aber für „Isolation“ keinen Mehraufwand, da er die Undo-Records schon für „Atomicity“ und „Consistency“ anlegen musste, und das DBMS diese damit ohnehin vorhandenen Metadaten hier einfach mitverwendet.

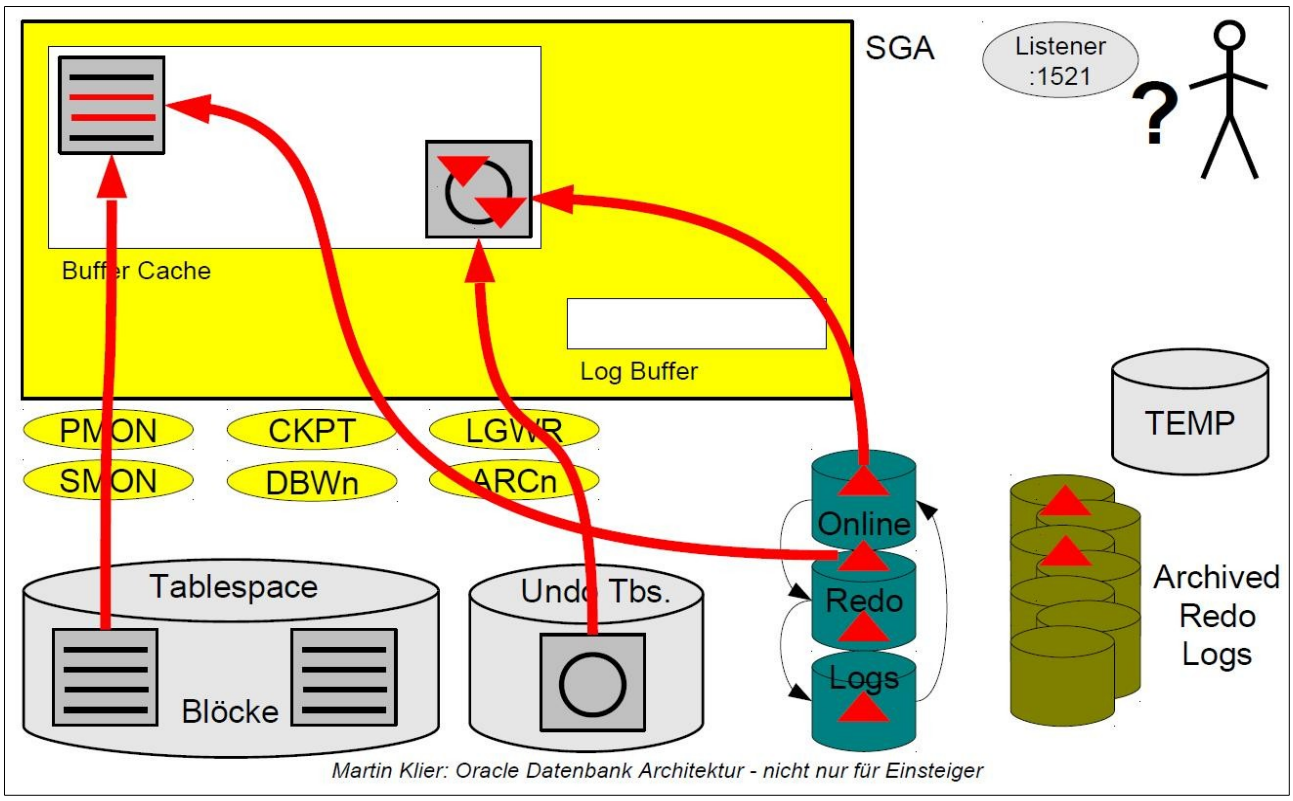
„Aufeinander gestapelte“ Transaktionsversionen sind zulässig, d.h. ein Consistent Read Clone kann durch wiederholtes Rollback durchaus mehrfach genutzt werden.

Konkurrierende Schreibvorgänge auf der gleichen Row durch zwei getrennte Transaktionen sind jedoch nicht möglich, der Erste sperrt die Zeile exklusiv, der Zweite und alle Folgenden müssen in einer Queue warten.

5.3 Crash Recovery: Das ganze Bild

Eine Änderung von Rows erzeugt Redo-Informationen, die für ein Crash Recovery benötigt werden. Auch die angelegten Undo-Records sichert das DBMS auf diese Weise ab, sie sind damit ebenfalls in den Online Redo Logs enthalten.





Im Fall eines Crashs sind zwar alle im RAM enthaltenen Informationen verloren, mit Hilfe der Online Redo Logs lassen sich aber Buffers und Undo-Records im Cache problemlos und ohne Benutzereingriff wiederherstellen.

Ein Sonderfall kann auftreten: Der DBWn könnte nicht commitete Buffer auf den Tablespace geschrieben haben. Diese werden ohne Ansehen des Transaktionsstatus beim Instance Recovery wiederhergestellt. Allerdings müsste dann in Zukunft jeder Leser den Weg über einen Consistent Read Clone gehen, um an den Stand vor der Transaktion zu gelangen. Die Session die die Transaktion eröffnet hatte, ist durch den Crash verloren und kann sie damit auch nicht zurückrollen. Hier greift dann der selbe Mechanismus wie bei jedem anderen Sessionverlust ohne Crash: einige Zeit nach einigen Sekunden rollt das System u.a. mit Hilfe des SMON-Prozess selbständig alle nicht commiteten Rows aller Tablespaces auf den letzten Zustand zurück. Das verursacht Last, und kann sich beim Wiederanlauf nach einem unbeabsichtigten Crash durchaus in beeinträchtigter Performance bemerkbar machen.

6 Die Basis für moderne Produkte

Viele der oben beschriebenen Punkte haben ihren Ursprung in „uralten“ Überlegungen von Oracle. Dennoch hat sich das beschriebene Design als recht flexibel erwiesen - viele der modernen Features (zum Teil sogar Alleinstellungsmerkmale) gründen sich auf das alte Konzept.

6.1 Data Guard (aka Standby-Datenbank)

Die Standby-Datenbank als Replikationslösung von einer Datenbank in eine andere stammt eigentlich nicht von Oracle, sondern hatte ihren Anfang in manuellen Skripten der Kunden-DBAs, die fertige Archived Redo Logs mit Betriebssystemmitteln kopierten und eine zweite Instanz im ständigen Recovery hielten. Später adaptierte Oracle dies als eigenes Produkt mit gleicher Funktion, wobei der ursprüngliche Ansatz bis heute unterstützt und sogar in der Lizenzierung gesondert berücksichtigt wird.

Allerdings kamen bald neue Funktionen dazu, die sich extern nicht verwirklichen ließen, die aber der Hersteller selbst einbauen kann. Zwischenzeitlich ermöglicht Dataguard (so der Marketingname

des Konzepts) damit eine Replikation ab dem Logwriter, d.h. ohne die Verzögerung und die Datenlücke die entstehen, wenn nur fertiggestellte Archive übertragen werden können.

Auf dem selben Weg wurde auch ein Two-Phase-Commit realisiert: Die Primärseite bestätigt auf Wunsch einen commit erst, wenn auch die Standby-Datenbank die Daten mindestens in ihre Redo Logs geschrieben hat. Damit ist sichergestellt, dass die Daten garantiert repliziert wurden.

6.2 Real Application Cluster (RAC)

Der Real Application Cluster („RAC“) in seiner heutigen Form existiert seit der Datenbankversion 10g, mit einem kleinen Vorläufer in der 9i. Er bietet als bislang einziges Datenbank-Cluster auf dem Markt uneingeschränkte Aktiv/Aktiv-Fähigkeiten, d.h. vollen, konkurrenten schreib/lese-Zugriff von mehreren Rechnersystemen auf die selbe Datenbank. Das geht einher mit einer relativ guten Skalierbarkeit, die allerdings sehr vom Lasttyp abhängt.

RAC basiert auf der Idee, dass sich zunächst jeder Rechner verhält, als wäre er die einzige Instanz für die Datenbank, wie oben ausführlich beschrieben. Oracle hat das Konzept dahingehend angereichert, dass z.B. eine RAC-Instanz zu ladende Buffers zunächst über ein Interconnect-Netzwerk aus dem Buffer Cache der Nachbarsysteme holt, bevor es Disk IO macht. Dies funktioniert auch mit Buffers, die dort zuvor manipuliert wurden, und noch nicht auf dem Massenspeicher liegen.

Konsistente Lesezugriffe im RAC funktionieren analog zum Verfahren auf einer Single Instance, jedoch können alle RAC-Instanzen die Undo-Tablespaces aller anderen Instanzen einsehen und daraus Undo-Records für die Erzeugung eines Consistent Read Clones gewinnen.

Im Crashfall sind alle Knoten eines RAC in der Lage, das im Bereich über das Redo-Verfahren beschriebene Instance Recovery und den im Undo-Verfahren erläuterte Rollback auch für andere, ausgefallene Knoten durchzuführen.

6.3 Flashback

„Flashback“ ist ein Marketingname, der mehrere verschiedene Technologien unter einem Begriff zusammenfasst. Die meisten davon basieren auf Undo, hier sollen nur zwei Beispiele genannt werden.

6.3.1 Flashback Database

Das Feature „Flashback Database“ erlaubt es dem DBA, eine ganze Datenbank zurückzurollen. Im Gegensatz zum klassischen Point-in-Time-Recovery, das das Zurückspielen eines Backups und anschließend einen Rollforward mit Hilfe von Archived Redo Logs erfordert, kommt Flashback Database ohne dieses zeitraubende Manöver aus.

Dazu wird die Instanz so konfiguriert, daß sie Undo-Records vor deren Überschreiben in sogenannten Flashback Logs auslagert. So weit diese Logs vorhanden sind, kann der DBA die gesamte Datenbank quasi „Transaktion für Transaktion“ bzw. „Row für Row“ zurück in die Vergangenheit versetzen.

Flashback Database ist vor allem bei großen Datenbanken interessant, wenn man nur relativ wenig zurück reisen möchte.

6.3.2 Flashback Query

„Flashback Query“ und „Flashback Version Query“ kommen auch ohne Flashback Logs aus, und kann den Inhalt des/der Undo-Tablespaces nutzen, so lange dieser nicht überschrieben wurde. Einfachste Abfragen erlauben dem User, seine Objekte „so wie sie zum Zeitpunkt X waren“

abzufragen (Flashback Query), oder sich auch die in einem Zeitraum liegenden Änderungen darstellen zu lassen (Flashback Version Query).

Beispiel Flashback Query:

```
SQL> select *
      from EMP
      as of timestamp to_timestamp(sysdate-8/24) ;
```

Beispiel Flashback Version Query:

```
SQL> select *
      from EMP
      versions between timestamp to_timestamp(sysdate-1/48)
            and to_timestamp(sysdate) ;
```

7 Autor

Martin Klier ist leitender Datenbankadministrator bei Klug GmbH integrierte Systeme (http://www.klug-is.de), dem Hersteller des Softwareproduktes iWACS und damit einem führenden Unternehmen für Intralogistiklösungen. Er verfügt über mehr als zehn Jahre Erfahrung in Aufbau und Betrieb von großen Oracle Datenbanksystemen. Nach den ersten Schwerpunkten auf RAC mit Dataguard (MAA) und integrierten Clustern kamen später Auslegung und Performanceoptimierung von OLTP-Systemen mit dynamisch wechselnden Lastprofilen hinzu.

Martin Klier bloggt Probleme und Lösungen auf http://www.usn-it.de und referiert regelmäßig bei Konferenzen wie COLLABORATE (USA), Oracle Open World (USA) und der DOAG Konferenz (EU) über spezielle Datenbankthemen.

Anschrift:
Martin Klier
Klug GmbH integrierte Systeme
Lindenweg 13
D-92552 Teunz
Telefon: +49 9671 9216-245
Fax: +49 9671 9216-6245

E-Mail martin.klier@klug-is.de
Internet: http://www.klug-is.de

Blog: http://www.usn-it.de