Oracle Database Concepts

م.عبد البديع مراد

RB Informatics; 25/10/2022

قواعد المعطيات المتقدمة

مقدمة:

تحدثنا في المحاضرة الماضية عن كيفية تشغيل oracle database وإطفائها وأن التشغيل يعني تشغيل oracle instance والتأكد من الملفات المهمة ثم فتحها للمستخدمين كما تعلمنا كيفية إطفائها وأخيرا تحدثنا عن الطرق التي من الممكن عبرها تغيير بارامتر أو أكثر من بارامترات الـ SPFile، في هذه المحاضرة سنتناول بعض المفاهيم النظرية المتعلقة بقاعدة بيانات أوراكل :

- Data Blocks, Extents, and Segments
- حالات التخزين في الـ data block
- Tablespace and data files
- Process architecture
- Dedicated mode and shared mode
- ملخص عملية التخزين

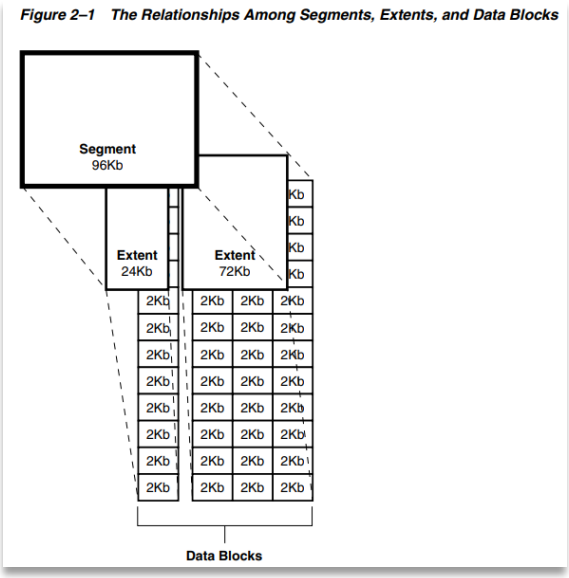
Data Blocks, Extents, and Segments:

يتم تخزين البيانات الموجودة في قاعدة بيانات أوراكل عن طريق وحدات تخزينية هي data blocks, extents, and segments وسنتعرف عليهم وعلى العلاقة بينهم.

تقوم أوراكل بتخزين البيانات في الـ data blocks وكل data block يكافئ عدد من الـ bytes من القرص وهي أصغر وحدة تخزين.

المستوى (level) التالي هو الـ extent وهو تخزين منطقي (logical) وليس فيزيائي والـ extent تكافئ عدد من الـ data blocks المتتالية المحجوزة لتخزين صنف معين من المعلومات.

Figure 2–1 The Relationships Among Segments, Extents, and Data Blocks



المستوى (level) التالي هو مستوى أكبر من الـ extent ويسمى segment، والـ segment هو مجموعة من الـ extents تحوي object والذي قد يكون جزء من الـ table

تنقسم قاعدة البيانات (Database) إلى وحدات تخزين منطقية تسمى Tablespaces، والتي تجمع Logical Structures معًا وسنتحدث أكثر عن الـ Tablespace في هذه المحاضرة.

نتيجة: كل tablespace يتألف من n data files وكل data file هو مجموعة من الـ segments وكل segment هو مجموعة من الـ extents وكل extent هو مجموعة من الـ data block.

Data Blocks:

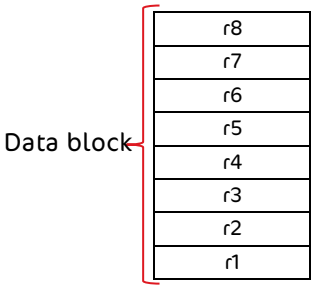
ليكن لدينا الجدول التالي T1 مكون من عدد من الأسطر $r_1, r_2, \dots, r_n$:

r_1	
r_1	
⋮	
r_n	

إن وحدة التخزين الرئيسية في أوراكل هي data block وإن قمنا بفتح ملف الـ SPFile سنجد أحد البارامترات هو db_block_size وحجمه (8 KB) تقريبا بشكل افتراضي. وهذه الـ data block هي أصغر وحدة تخزين في أوراكل في الـ data file (على الـ hard disks) ويتم تخزين الـ data داخلها.

حالات التخزين في الـ data block

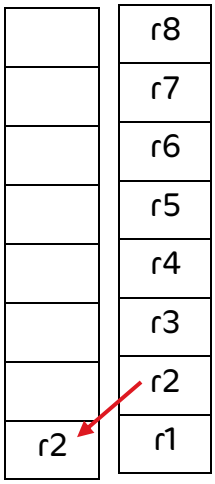
الحالة الأولى: في حال كان حجم السطر (r) في الجدول (t) أصغر من حجم الـ data block، فبفرض كان حجم السطر (record) هو 1 KB فإن الـ data block الواحدة ستتسع لما يقارب 8 أسطر وإذا افترضنا وضع 8 أسطر في هذه الـ data block ستكون ممتلئة تماما.



في حال أردنا تعديل أحد الأعمدة (c) من أحد هذه الأسطر ولنفرض أنه عمود يحوي نص وقمنا بزيادة حجم النص بشكل ملحوظ كالتالي:

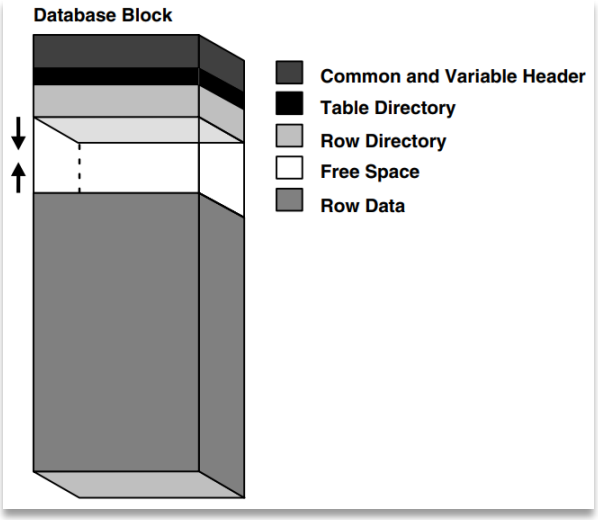
```
Update t1
Set c2 = "characters .... ...."
Where r = r2
```

إن هذا التغيير أدى إلى زيادة حجم السطر r2 فأصبح أكبر من 1KB، وبالتالي لن يتمكن من تخزين السطر بعد التعديل في الـ data block التي كانت موجودة فيه لأنها كانت ممتلئة بالفعل وهذه الزيادة في الحجم لن تتسع في الـ data block.



في هذه الحالة سيتم نقل السطر r2 الذي زاد حجمه إلى data block أخرى، ولكن لن يتم تحرير مكانه القديم بل سيبقى محجوز وسيحوي على مؤشر لمكانه الجديد. وهذا المؤشر هو الـ row id. وتتكرر هذه العملية في كل مرة يتم فيها التعديل على سطر بحيث يزيد هذا التعديل من حجم السطر ويصبح حجمه أكبر من المساحة المتاحة ضمن الـ data block (أي من الممكن تواجد مؤشر على مؤشر ....). تسمى عملية نقل البيانات من data block لأخرى بتهجير البيانات (data migration).

نلاحظ أن عملية تهجير البيانات عملية مكلفة لأنها تقوم بحجز مساحة تخزين إضافية غير مفيدة، لذلك لا تقوم أوراكل بتخصيص كامل الـ data block لتخزين الـ data بل تقوم بتقسيمه إلى أكثر من مقطع كما يظهر في الشكل:



قسم Header (Common and Variable): ويحوي على معلومات عامة عن الـ block مثل عنوان الـ block ونوع الـ segment (مثلا data أم index).

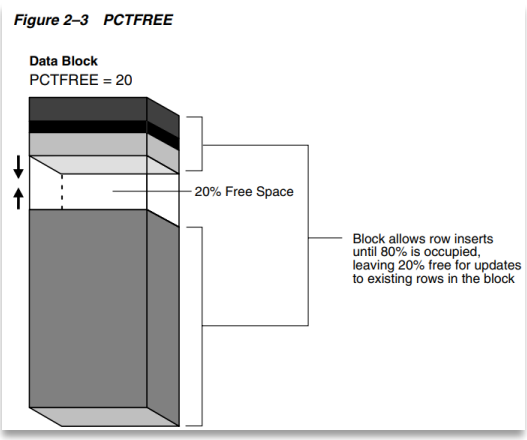
قسم Table Directory: يحتوي هذا الجزء من كتلة البيانات على معلومات حول الجدول الذي يحتوي على صفوف في هذه الـ block.

قسم الـ row directory: يحتوي هذا الجزء من كتلة البيانات على معلومات حول الصفوف الفعلية في الكتلة

قسم الـ row data: يحتوي هذا الجزء على بيانات جدول أو فهرس.

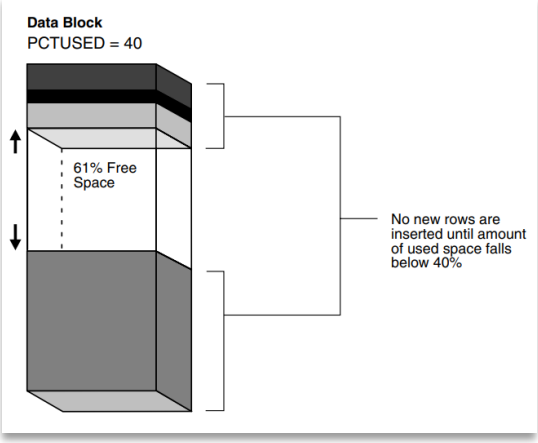
قسم الـ free space: يتم تخصيص مساحة خالية لإدراج صفوف جديدة ولتحديثات الصفوف التي تتطلب مساحة إضافية لتقليل عملية تهجير البيانات (على سبيل المثال، عند تحديث قيمة null إلى قيمة غير null)

من الممكن التحكم بالـ free space بشكل تلقائي أو يدوي فمن الممكن على سبيل المثال (تحديد المساحة القابلة للاستخدام بـ 80%)، إذ يوجد بارامتران يمكن التعديل عليهما هما (PCTFREE, PCTUSED) وهي physical parameters يمكن تحديدها عند إنشاء schema object أو تغييره



PCTFREE: يحدد هذا البارامتر النسبة المئوية الدنيا لحجز data block كمساحة خالية للتحديثات المحتملة للصفوف الموجودة بالفعل في تلك الكتلة. فمثلا، بفرض قمنا بتحديد PCTFREE في تعليمة CREATE TABLE بقيمة 20 بالمئة: PCTFREE 20

ينص هذا على أن 20% من كل data block في الـ segment التابع لهذا الجدول تكون ضمن الـ free space ومتاحة للتحديثات المحتملة للصفوف الموجودة بالفعل داخل كل data block.



PCTUSED: يحدد هذا البارامتر النسبة المئوية الدنيا للـ block التي يمكن استخدامها للـ row data.

بعد ملء data block إلى الحد الذي تحدده PCTFREE، تعتبر Oracle أن الكتلة غير متاحة لإدراج صفوف جديدة حتى تصبح النسبة المئوية لتلك الـ block تحت قيمة الـ PCTUSED.

حتى يتم تحقيق هذه القيمة، تستخدم Oracle المساحة الخالية من كتلة البيانات فقط لتحديثات الصفوف الموجودة بالفعل في كتلة البيانات.

على سبيل المثال، بفرض قمنا بتحديد البارامتر كالتالي CREATE TABLE: PCTUSED 40

في هذه الحالة، تعتبر الـ data block المستخدمة لـ data segment من الجدول غير متاحة لإدراج أي صفوف جديدة حتى تنخفض مساحة المساحة المستخدمة في الكتلة إلى 39% أو أقل (بافتراض أن المساحة المستخدمة للكتلة قد وصلت سابقًا إلى PCTFREE).

الحالة الثانية: في حال حجم الـ row الواحد أكبر من حجم الـ data block فمثلا كان حجم الـ data block هو 8KB وحجم الـ row هو 20KB عندها يتم تخزين الـ row على أكثر من data block (n data block) تسمى هذه العملية بـ Row Chaining وهي عبارة عن سلسلة (chain) من الـ data blocks

ولكن إذا أردنا إجراء query كالتالي على سبيل المثال:

$$select\ *$$
$$from\ t_1$$
$$where\ r = r_1$$

فكيف ستستطيع أوراكل معرفة أن التخزين قد تم على أكثر من data block؟ لهذا السبب يوجد أكثر من مستوى في التخزين فالوحدة الأساسية في التخزين هي الـ data block وكل مجموعة من الـ data block محزمة بطبقة تسمى extent.

تذكرة:

ينصح بأن يكون حجم الـ row صغير وبناء أكثر من جدول بدلا من أن يكون حجم الـ row كبير وذلك للتقليل من الـ chaining، ولكن لا بأس بالقيام بذلك في الحالات الضرورية فأوراكل يجمع مجموعة data blocks في extent كما ذكرنا، فلابأس بحدوث chaining.

مشكلة الحالة الأولى:

لننظر إلى الوضع التالي في حال قمنا بتخزين صورة أو ملف حجمه n MB ضمن الـ data blocks. ففي حال قمنا بالتعديل (update) على الصورة بحيث زاد حجمها ولم تتسع لها الـ data blocks عندها سيقوم بإجراء migration للصورة إلى مجموعة data blocks أخرى، ولكن الـ data blocks التي كانت محجوزة من قبل الصورة أو الملف لن يتم تحريرها كما ذكرنا في شرح الحالة الأولى بل ستحوي مؤشرات إلى الـ data blocks الجديدة التي أصبحت تحوي الصورة أو الملف. وتكمن المشكلة في ذلك مع تكرار هذه العملية سيتضخم في الـ tablespace بشكل سريع دون وجود كمية بيانات مكافئة لهذا الحجم.

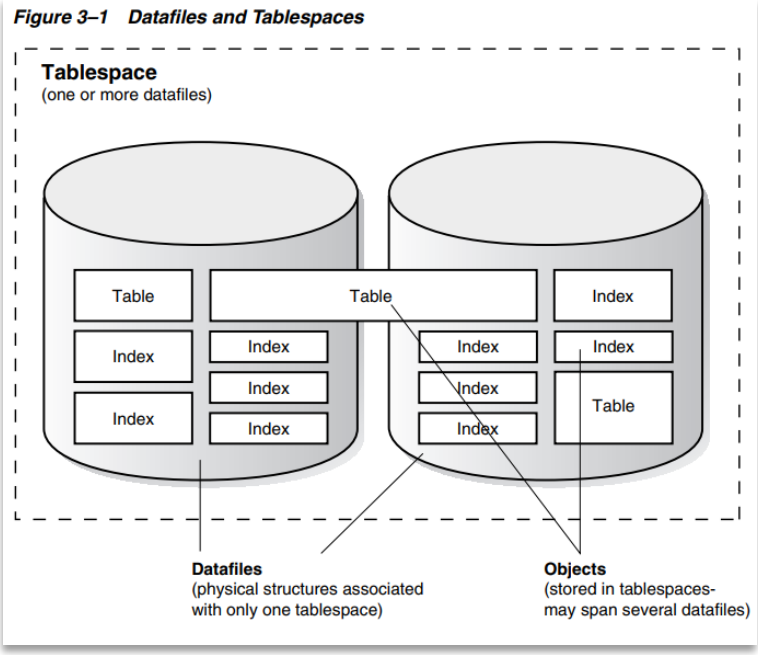
الحل لهذه المشكلة هو أخذ صورة عن الـ data base (export) ثم إجراء import للبيانات وذلك كل فترة، فبإجراء import للبيانات لا يعود إلى حجز data blocks من أجل مؤشرات بل يقوم بإعادة بناء الـ structure بشكل صحيح ويقوم بتخزين البيانات في data blocks جديدة، ونحتاج لهذا الحل عندما توجد عمليات DML كثيرة على rows لها حجم كبير.

:Tablespaces and Datafiles

عندما نقوم ببناء الـ database لا نتعامل مع الـ data blocks والـ extents والـ segments بل نتعامل مع الـ tablespace كوحدة تخزين. وبالتالي أثناء عملية بناء قاعدة المعطيات عندما نريد تخزين الـ objects نحتاج إلى بناء طبقة (layer) هي طبقة الـ table space وهي طبقة منطقية (logical) ليست فيزيائية مبنية على الـ datafiles (والـ datafiles هي وحدات تخزين فيزيائية موجودة على القرص).

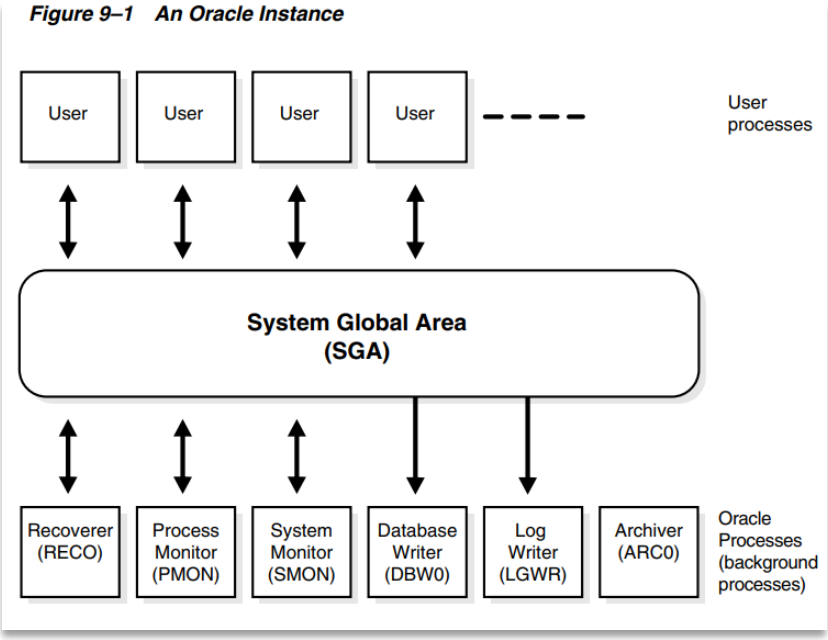
وقد وجدت طبقة الـ tablespace لتشمل datafile واحد أو عدة datafiles لأنه من المحتمل (كما يظهر في الصورة) وجود tables ممتدة على أكثر من datafile وبالتالي وجدت هذه الطبقة لتسهيل عملية الـ management للـ datafiles وتخفيف عبئ عن الـ administrator

Figure 3–1 Datafiles and Tablespaces



Process architecture:

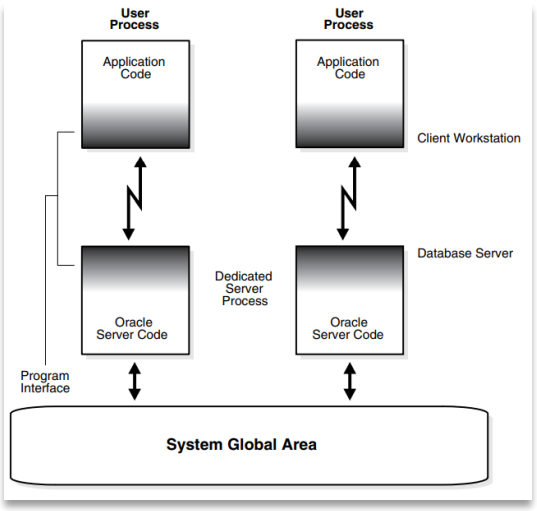
ذكرنا في المحاضرة السابقة أنه لتشكل الـ oracle instance يتم حجز system global area (SGA) وتشغيل عدد من الـ background processes



إذا أراد أحد الـ user القيام بـ connect على قاعدة المعطيات فهذا يعني أنه يقوم بفتح session.

هنا تدعم oracle نوعين من الاستجابة هما:

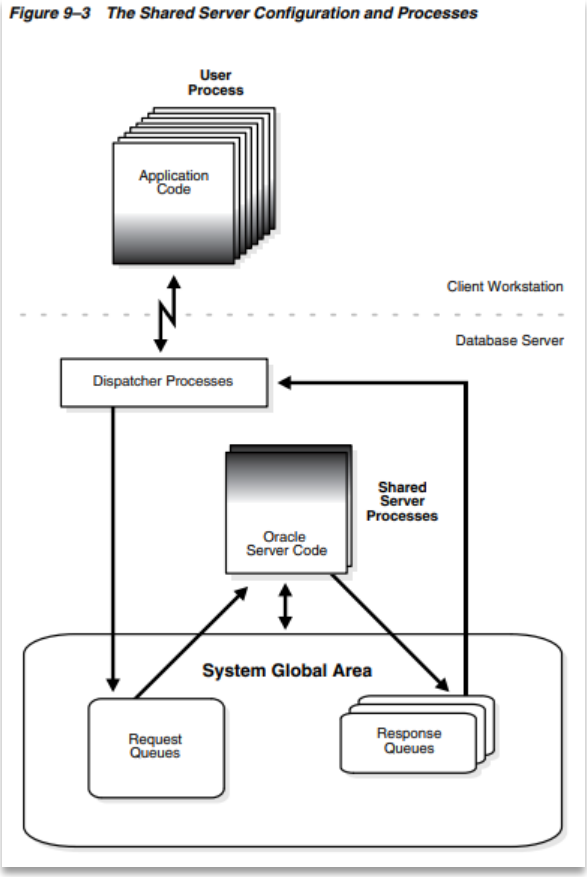
1. Dedicated mode
2. Shared mode



1) في حالة الـ **dedicated mode**، كل session يقوم المستخدم بفتحها؛ يقابلها user process تبنيها oracle لتخديم الـ user واستقبال الـ requests التي يقوم الـ user بطلبها ويرسل له الـ responses ولكن هذا النوع من الاستجابة غير مقبول إذ يمكن لأحد المستخدمين فتح عدد كبير من الـ sessions دون استخدامهم وبالتالي يقوم بحجز الكثير من الموارد (ذاكرة ومعالجة) قد يؤدي هذا الاستهلاك لعدم تمكن أوراكل من إنشاء process **user** عندما يريد **user** آخر فتح session (connect)، ويحدد عدد الـ processes الممكن فتحها من الـ SPFile إذ يوجد بارامتر processes يسند له عدد الـ process التي يمكن لأوراكل فتحها. وإذا وجد sessions غير فعالة فإن أوراكل لا تقوم بإغلاقها بل يجب أن يقوم المستخدم بذلك، لذلك لا يستخدم dedicated mode في الحياة العملية

2) shared mode

في هذا ال mode ايضا يوجد user processes مقابلة للـ connections ولكن لا تكون كل process مقابلة لـ connection واحدة بل قد تكون process تقوم بتخديم أكثر من connection كما انه يتم التحكم بال processes من قبل dispatcher فأصبح تواصل المستخدم مع الـ dispatcher وليس مع الـ user processes مباشرة ثم يقوم ال dispatcher بالتواصل مع ال oracle processes اذا يقوم الـ dispatcher بإرسال الـ requests للـ system global area فيقوم الـ oracle process بتحضير البيانات وتحضير الـ response ثم يرسله إلى الـ dispatcher ليقوم الـ dispatcher بإرسال الـ response للمستخدم الذي قام بإجراء الطلب المتوافق مع الـ response الذي يريد إرساله وبالتالي الـ shared mode هو أفضل من ناحية إدارة الموارد في oracle فقد يوجد 200 processes تخدم حوالي 1000 sessions أو أكثر.



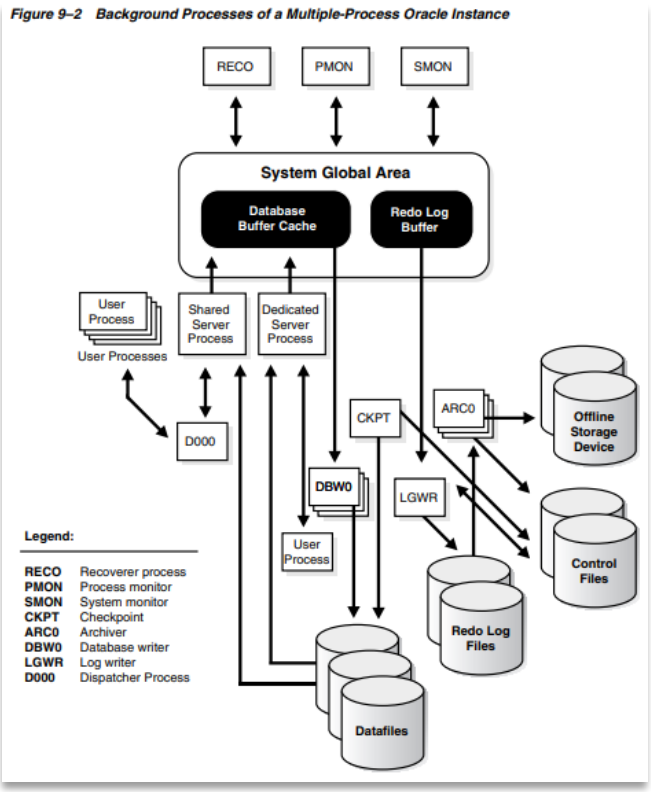
وسنتحدث في الدرس القادم عن إمكانية إعطاء life للـ sessions بحيث إذا بقيت غير فعالة لمدة معينة يتم إجراء kill session.

الصورة التالية تظهر ملخص عملية التخزين كاملة:

ذكرنا في الدرس الماضي أن أوراكل لديها أربع أنواع مهمة من الـ files هي

datafiles, SPFILE, control files, log files

لدينا الـ user processes في البداية والتي تعبر عن وجود sessions مفتوحة يقوم المستخدم عبرها بتعليمات DML ثم تتواصل هذه الـ user processes مع الـ dispatcher (يعتبر هذا المخطط أن أوراكل تستخدم الـ shared mode) ويتواصل الـ dispatcher مع الـ oracle processes التي تتخاطب مع الـ system global area ونلاحظ أن جلب البيانات يتم من الـ memory أما في حال لم توجد البيانات في الذاكرة فيتم جلبها من الـ disk ونقلها إلى الـ memory



عن طريق الـ DB reader، وفي حال كانت التعليمة هي تعليمة insert أو update أو delete فتقوم اوراكل بكتابتها على الـ memory وتكون إمكانية الكتابة على الـ memory محددة بحجم معين (عبر البارامترات shared pool, large pool,... الموجودة في الـ SPFILE) وعندما يمتلئ الحجم المحدد لتخزين الـ insert, update, delete في الـ memory يتم نقلها إلى الـ disk عن طريق الـ DB writer ليتم كتابتها على الـ datafiles في الـ disk.

أيضا نلاحظ من الصورة وجود buffer log ومرتبطة مع الـ log writer. إذ أن أوراكل تدعم نوعين من الـ backup وهي ARCHIVELOG mode و NOARCHIVELOG mode فإذا كانت أوراكل تعمل في وضعية الـ ARCHIVELOG mode فإن كل ما تم كتابته من تعليمات insert update, delete يتم إجراء تخزين له (بشكل مكافئ للتعليمات) على hard خارجي من خلال الـ archiver لأنه في الـ archive mode نقوم بتخزين كل الـ logs التي تتولد أثناء التعليمات على الـ log file بحيث عندما يمتلئ أحدها تقوم بنقله على الـ hard حتى يُتمكن من استرجاع قاعدة البيانات في حال الانهيار، أما في حالة الـ non archive mode لا يتم تخزين الـ log files على الهارد وإنما يقوم بملأ الـ log files جميعها (هي 3 عادةً) فإذا امتلأت جميعها يقوم بمسح محتوى الـ log file الأول ويعيد الكتابة فيه ولا يقوم بنقل محتوياتهم إلى الـ hard وفي حال أردنا أخذ نسخة عن الداتا يتم أخذ نسخة عنها من الـ data files عن طريق إجراء export, import (أخذ image عن الداتا).

نلاحظ في الصور أيضا وجود ما يسمى checkpoint وهو يقوم بمعرفة الـ transaction التي يتم تنفيذها على الـ data file ويقوم بتخزين رقم آخر transaction تم تنفيذه على الـ data في الـ control file ويقوم بذلك حتى يتم مطابقة بين الـ two sequences ففي حال حدوث تطابق عندها لا تحتاج أوراكل لإجراء recovery للـ database أما إذا وجد اختلاف بين آخر transaction مطبق على الـ database (الـ serial number الخاص به) والرقم المخزن في الـ control file عندها يعرف الـ database optimizer أن قاعدة المعطيات تحتاج لإجراء recovery.

تم بعونه تعالى..♡