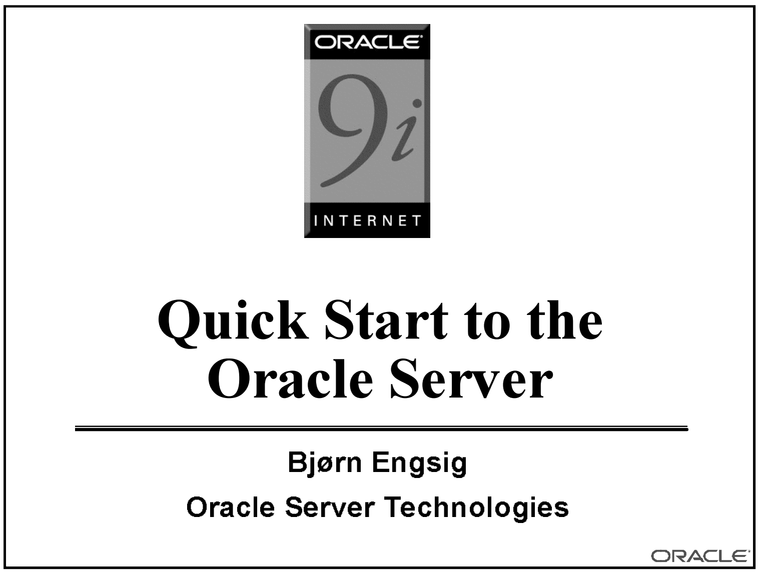


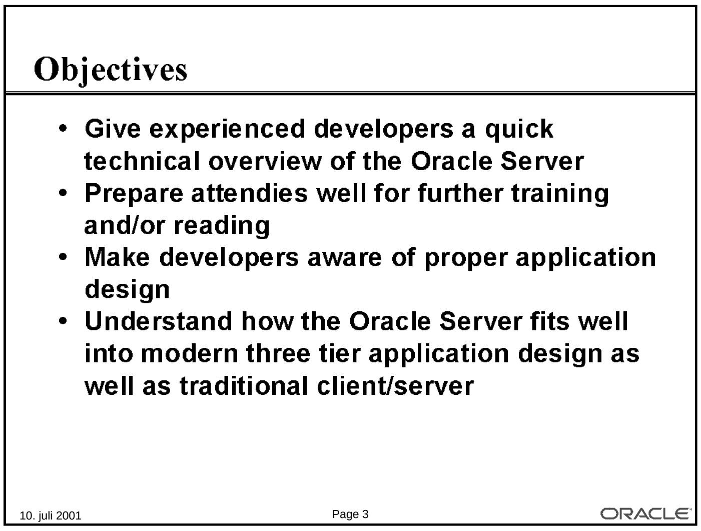
Quick Start to the Oracle Server

Bjørn Engsig

Oracle Server Technologies

This briefing is a quick introduction to the Oracle9i Server. The objective is to give attendees, who typically are skilled application designers with knowledge of other database products, a quick overview of the components of the Oracle9i server, but still with high technical contents.

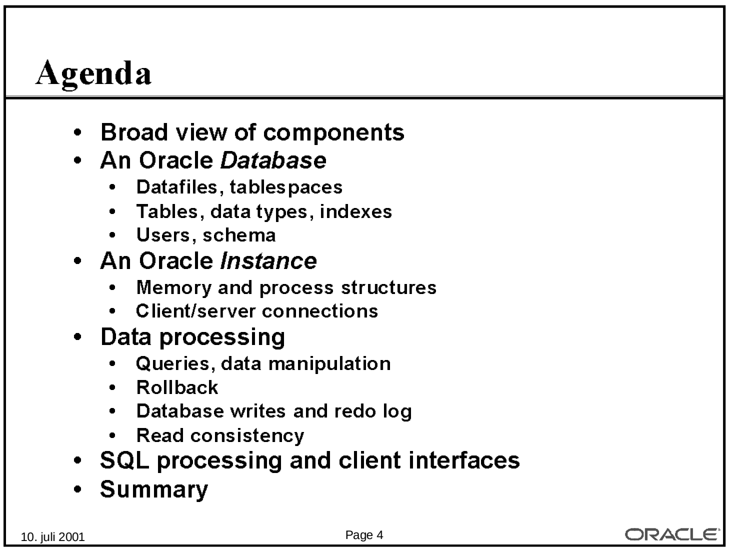
At the end, attendees should be able to choose further reading or training to start development projects using the Oracle database.



Objectives

- Give experienced developers a quick technical overview of the Oracle Server
- Prepare attendies well for further training and/or reading
- Make developers aware of proper application design
- Understand how the Oracle Server fits well into modern three tier application design as well as traditional client/server

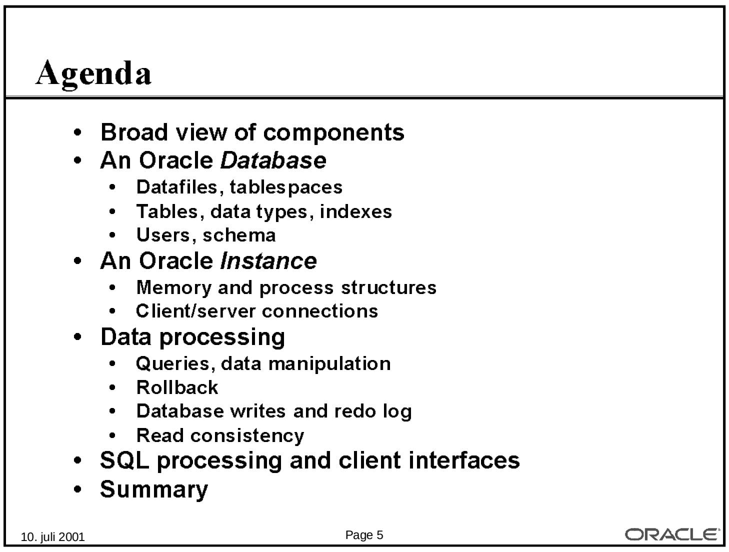
10. juli 2001 Page 3 ORACLE



Agenda

- Broad view of components
- An Oracle *Database*
  - Datafiles, tablespaces
  - Tables, data types, indexes
  - Users, schema
- An Oracle *Instance*
  - Memory and process structures
  - Client/server connections
- Data processing
  - Queries, data manipulation
  - Rollback
  - Database writes and redo log
  - Read consistency
- SQL processing and client interfaces
- Summary

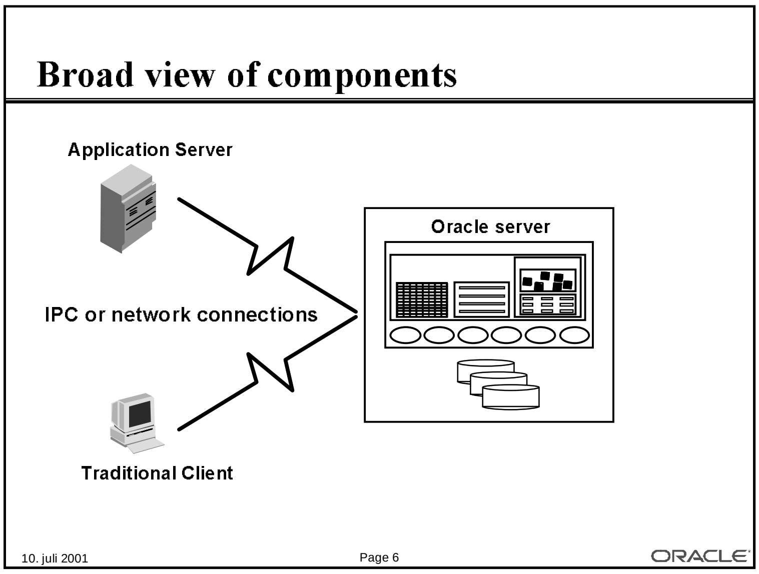
10. juli 2001 Page 4



Agenda

- Broad view of components
- An Oracle *Database*
  - Datafiles, tablespaces
  - Tables, data types, indexes
  - Users, schema
- An Oracle *Instance*
  - Memory and process structures
  - Client/server connections
- Data processing
  - Queries, data manipulation
  - Rollback
  - Database writes and redo log
  - Read consistency
- SQL processing and client interfaces
- Summary

10. juli 2001 Page 5 ORACLE



Broad view of components

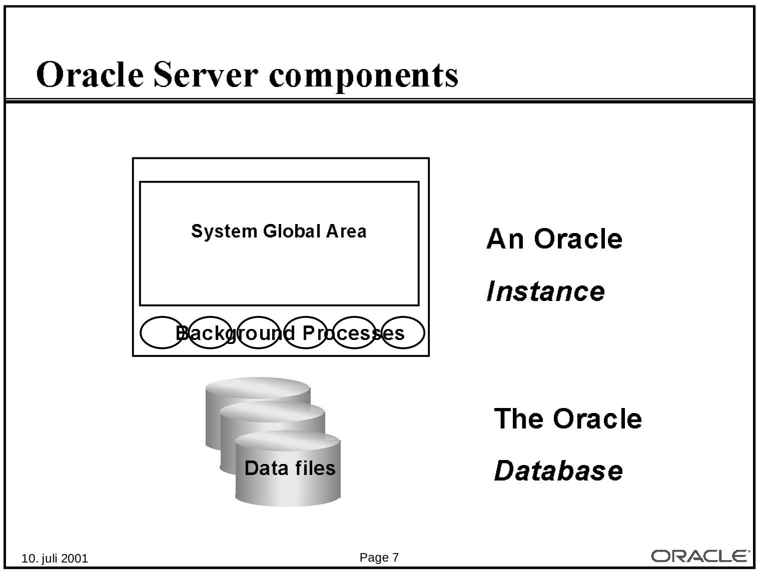
10. juli 2001 Page 6 ORACLE

The broad view of an Oracle Server and the related components show:

• The *Oracle Server* itself, which is were the database resides. In this picture, the server is simply considered a "black box", which most of the rest of this presentation looks inside.

• Systems connected to the Oracle Server. This can be either traditional *clients*, directly connected to the server, or it can be *Application Servers*, which run some application code, and to which clients connect (not shown). In this entire presentation, the term *client* is used throughout, as the behaviour of a traditional clients and an application server is the same, when seen from the database server.,

• *Connections*, which are some mechanism by which different processes can communicate. In practice, this is either physical network connections, often using TCP/IP, or it can be some inter process communication, e.g. using Unix pipes, when the client and the database server are actually executing on the same computer. In this presentation, there is no explicit distinction between a connection over a physical network, and one using inter process communication.



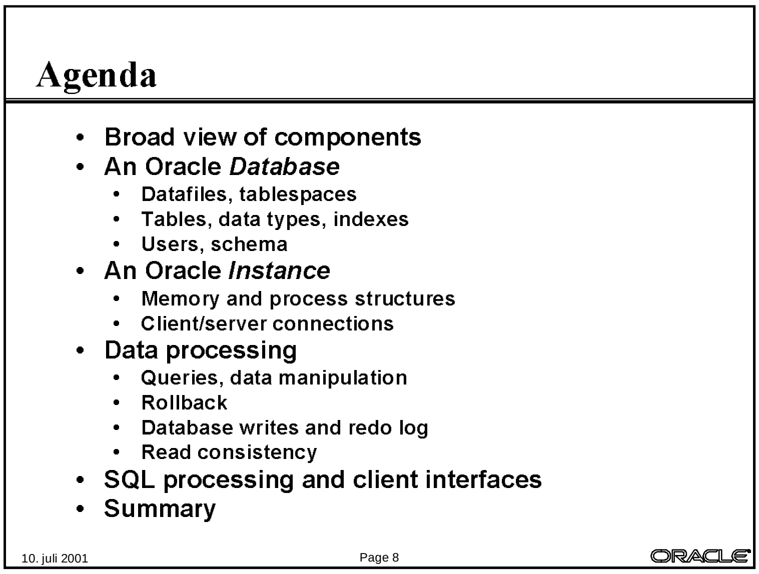
Oracle Server components

Looking into the Oracle server, two major components are seen:

• The Oracle *Database*, which is the set of operating system files, where the application's actual data are stored (such as customers, orders and order lines), together with Oracle's own internal information (such as information about users registered in the database, or tables found in the database). The Oracle Database as such, cannot be directly accessed in any way (except for backup) by clients. In contrast to other systems, there is no corelation between the concept of a user, an application or a file system and an Oracle database; rather, there is often only one database on each server, and you can even have a single database spanning multiple servers. This is known as Oracle Real Application Clusters.

• The Oracle *Instance*, which is a set of memory and process structures, running on a specific computer. This is the point of access for clients, and the instance is responsible for translating the SQL calls given by the client, to acual data transfers to and from the database stored in the operating system files. An Oracle instance is normally associated with an Oracle database, and those two together make up the Oracle Server. However, an Oracle instance may exist without being associated with an Oracle database; this is e.g. the case before the database is created or as an intermediate step during the startup of the Oracle server. An instance cannot be associated with more than one database, but one database (on a set of physical disks), can be associated with multiple instances (each on a separate computer), if the actual hardware configuration allow multiple computers to concurrently access a single set of disks.

• Inside the instance, two important parts are seen: The *System Global Area* (or *SGA*), which is a shared memory segment, typically of a size, which is roughly half of the physical RAM available on the computer. There is no direct access from clients to the SGA. Additionally, there is a set of *background processes*, which are working indirectly on behalf of the clients to do various specialized tasks, such as clean-up. These background processes are directly attached to the SGA, and can directly read and write to the disk files making up the database.



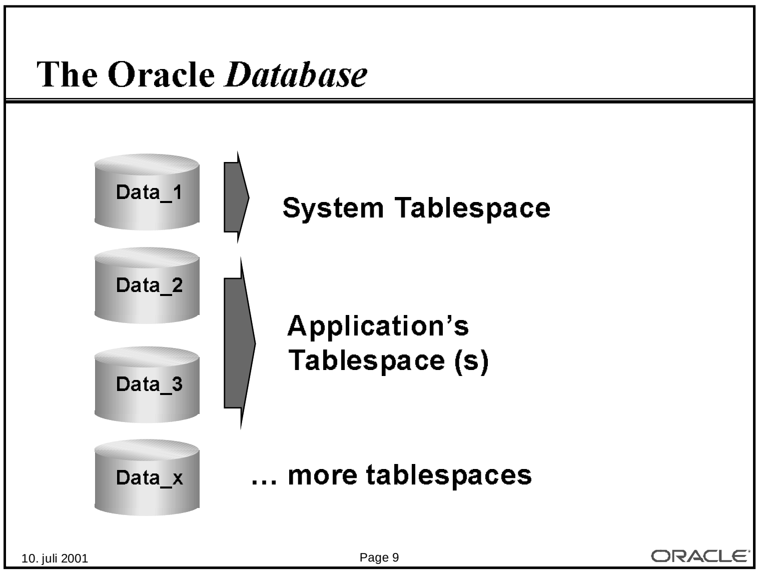
Agenda

- Broad view of components
- An Oracle *Database*
  - Datafiles, tablespaces
  - Tables, data types, indexes
  - Users, schema
- An Oracle *Instance*
  - Memory and process structures
  - Client/server connections
- Data processing
  - Queries, data manipulation
  - Rollback
  - Database writes and redo log
  - Read consistency
- SQL processing and client interfaces
- Summary

10. juli 2001 Page 8 ORACLE

An Oracle *Database* is the real set of files that make up your database. Remember, that there is normally a single database on each server, and the Oracle database will therefore normally be by far the application, that uses most of the available diskspace. The database itself contains both Oracle's own internal data, referred to as the Oracle *data dictionary*, and it contains the actual user data of one or more applications.

All data stored in the operating systems files, that make up the Oracle database, can only be accesed using the Oracle software. The only excpetion to this, is that backup of the Oracle database can be done by directly creating copies of the files making up the database.

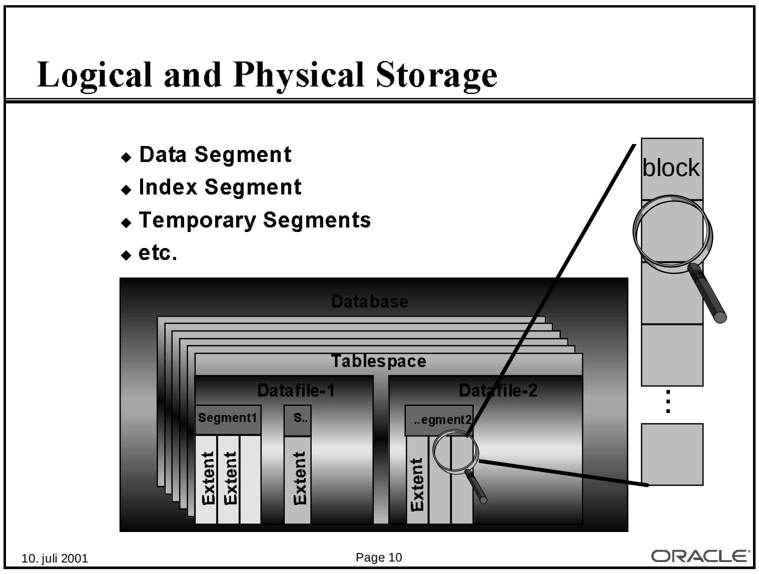


The Oracle *Database*

• All data of the database are stored in *Tablespaces*, which can be viewed as logical disks. Each tablespace is made of one or more physical disks, and can have sizes from megabytes to terabytes.

• Oracle uses a set of standard tablespaces, including the *system* tablespace, which is where Oracle stores its own internal information about e.g. users, tables and columns. Additionally, there are standard tablespaces used for e.g. temporary storage, etc. Only the tablespace with the actual name, *system*, is found right after database creation, other tablespaces, including the system related ones, are created subseequently.

• Each application (or set of related applications) define one or more tablespaces for storing that application's data. The names of these can be chosen freely, and can e.g. reflect application type such as *manufactoring, finance*, or you can have tablespace names reflecting use such as *appl_data*, and *appl_index*.



Logical and Physical Storage

- Data Segment
- Index Segment
- Temporary Segments
- etc.

10. juli 2001 Page 10 ORACLE

Tablespaces are the logcal storage media of an Oracle Database. Each tablespace contains data from one or more *segments*, such as the rows of a table, or the index of a table, and each *segment*, is made up of one or more *extents*.

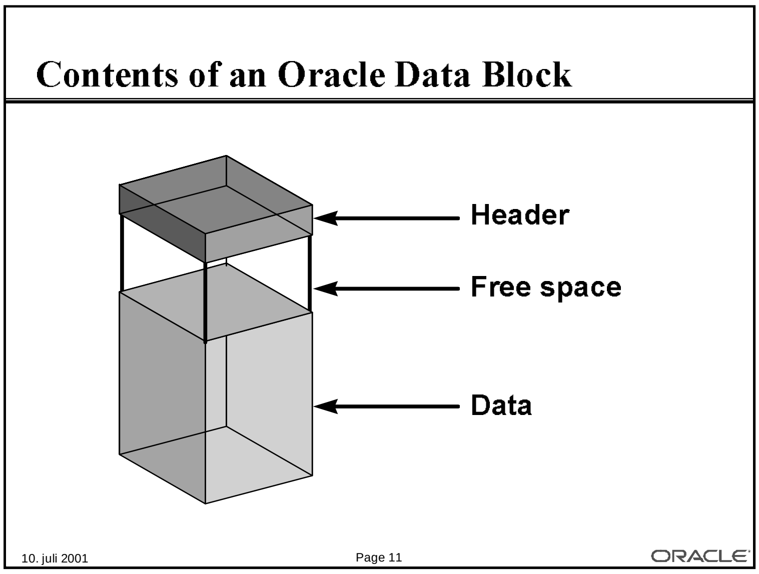
• The picture shows a tablespalce, that is made up of two physical *data files*.

• There are two segments shown, the yellow one (lightest grey) is made up of three extents, and the pink (medium grey) contains four extents. These could e.g. be a table and one of its indexes. The remaining part of the tablespace are unused blocks.

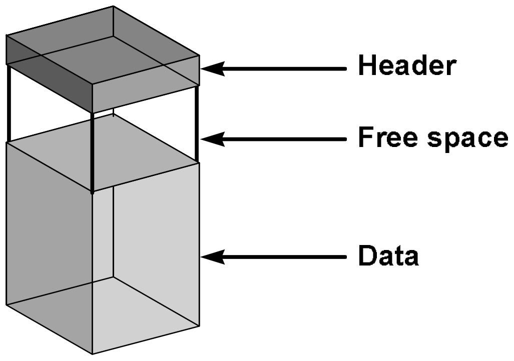
• A segment, including an initial extent, is created when you create an *object*, such as a table or an index, and more extents are added as necessary, when data is inserted.

• Extent size can be controlled at the tablespace level, for each segment individually, or by explicitly allocating an extent of a certain size. Normally, extent sizes are controlled by Oracle, by using a bitmap of all used of free extents. Alternatively, extent size and allocation can be managed by the database administrator.

• Note, there is *not* a one-to-one mapping between tables or indexes and datafiles, although you can specify, that a specific table is the only one stored in a specific tablespace.



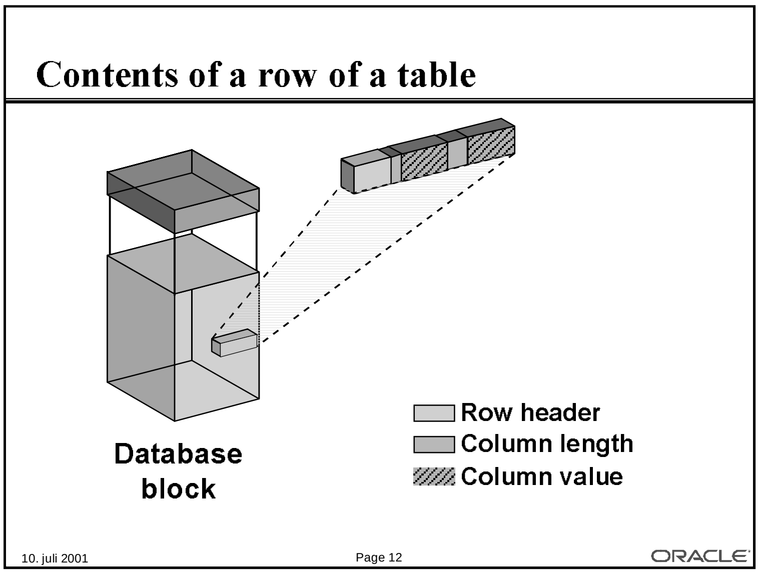
Contents of an Oracle Data Block



10. juli 2001 Page 11 ORACLE

All data is stored in Oracle blocks, the size of which are defined when the database is created. Typical sizes are 4kb, 8kb, or 16kb.

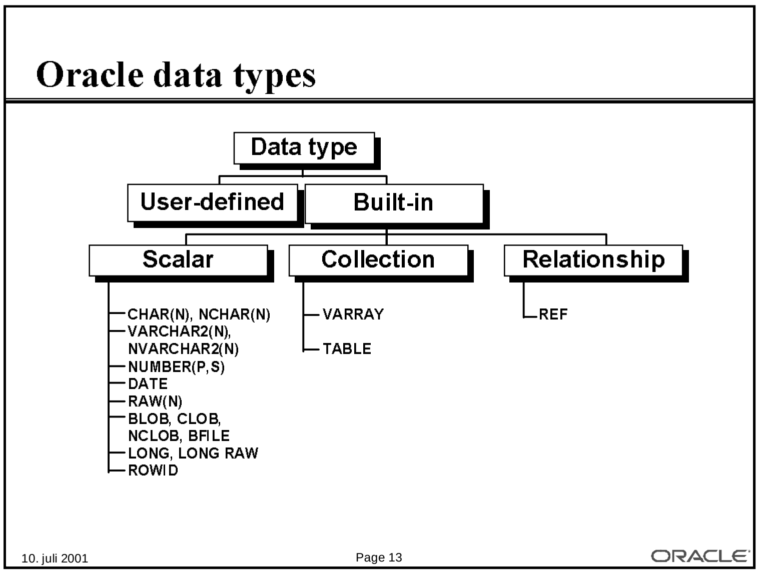
Each *block* contains a header, which includes a directory of rows, and it includes the actual data. Space management can be applied by the database designer and/or database administrator to control the amount of free space in each block, that is used to insert new rows or to update existing rows.



Contents of a row of a table

10. juli 2001 Page 12 ORACLE

Each *row* of a table is stored in the database block. The row storage includes a header and subsequent colun length/column data pairs. A row is completely identified by its *rowid* which consists of database file number (mapped to file name), block number in that file, and row number within the block.



Oracle data types

10. juli 2001 Page 13 ORACLE

Oracle data types are:

char, nchar - Fixed length character string, maximum 2000 bytes, but only recommended for smaller strings.

varchar, varchar2, nvarchar, nvarchar2 - Variable lenght character string, maximum 4000 bytes. The “2” types are currently identical to the normal types, but the latter may change with an evolving SQL standard.

number - Variable lenght numbers, scale and precision can be specified. The datatype is 100% identical on all Oracle platforms, which would not be the case with *native* types such as **integer**, or **float**. When declaring tables, native types are mapped to corresponding number types. The maximum precision is 38 decimal digits, and the maximal range is at least $\pm 10^{125}$.

date - Fixed length date (7 bytes) with second resolution.

rowid - Row identifier, guaranteed to be unchanged during the lifetime of a row.

raw - Raw binary data, maximum 2000 bytes, can be indexed.

clob, nclob, blob- Text or binary large objects, maximum 4Gb, cannot be indexed.

long, long raw - Text or binary long data, maintained for compatibility with older versions only.

The normal text datatypes (i.e. without “n”) are stored in the database character set, which must have the first 127 ASCII characters (or all printable EBCDIC characters) at their usual location; this is e.g. the case with the variable length Unicode character set. The text datatypes with “n” (nchar, nvarchar(2), nclob) are stored in the database’s national character set, which can use any character set e.g. fixed width multi-byte character sets. If the client and the server use different character sets, the underlying interface will translate between them.

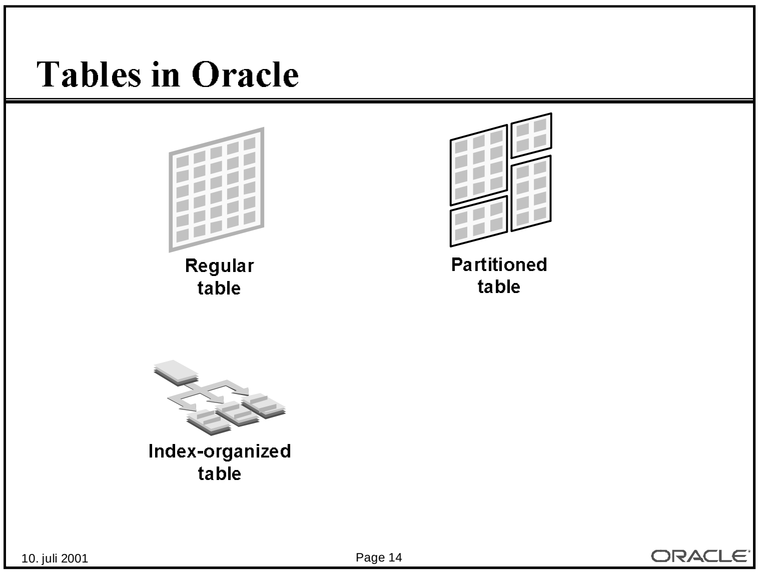
Extensive implicit or explicit data conversion is possible.

In the object relational model, the following extra date types are available

table of, varray - Collections, i.e. table or varying arrays of other types.

user defined types - Records of scalars or other types

ref - References to objects



Tables in Oracle

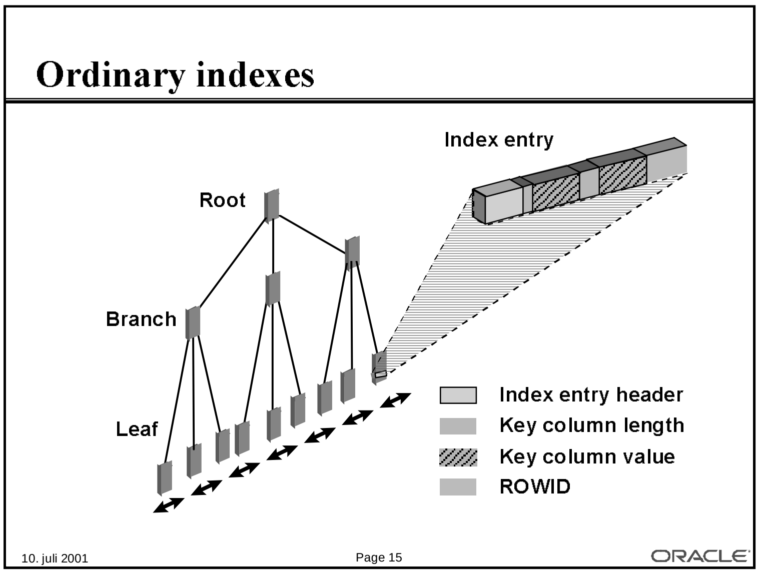
Regular
table
Partitioned
table
Index-organized
table
10. juli 2001
Page 14
ORACLE

Tables in Oracle can be stored in three different ways:

• Ordinary tables are stored with all rows in no particular order. In most cases, one or more indexes will be used as well.

• In *Partitioned* tables, rows of the table are stored in different tablespaces depending on a *partition key*, e.g. one partition per month of data or one partition per geographical region. This is typically used with very large databases (100+ Gb), and does e.g. allow database administrators to backup (and recover) parts of a table at different times.

• In *index organized tables*, an index (typically the primary key index) and the rows of the table are stored together. This implies a faster access using the index. Multiple indexes can still be used with this table storage.

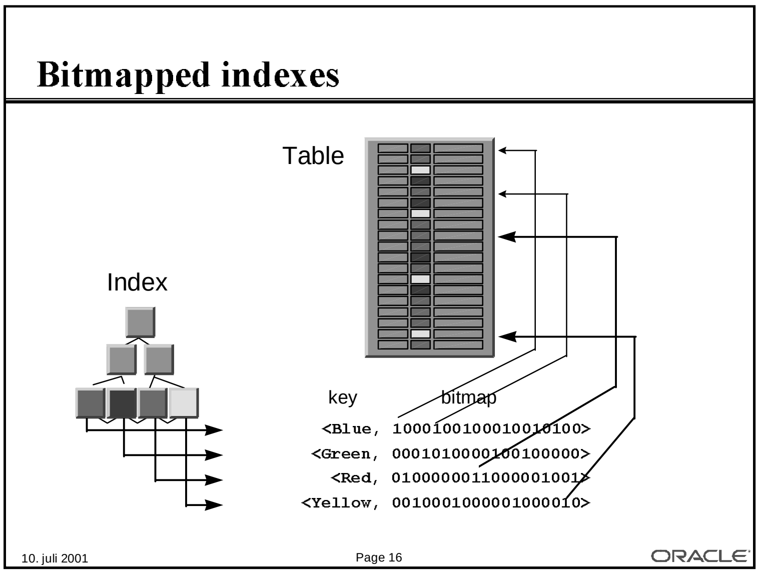


Ordinary indexes

- ▭ Index entry header
- ░ Key column length
- ▨ Key column value
- ░ ROWID

10. juli 2001 Page 15 ORACLE

Ordinary *indexes* are binary an contains one or more levels of *branch* blocks (the top one being the *root*), and one level of *leaf* blocks. Each index entry stores a header, the actual column values and a rowid pointing at the data block. In case of an index organized table, the rowid part is replaced by the full row of the table.

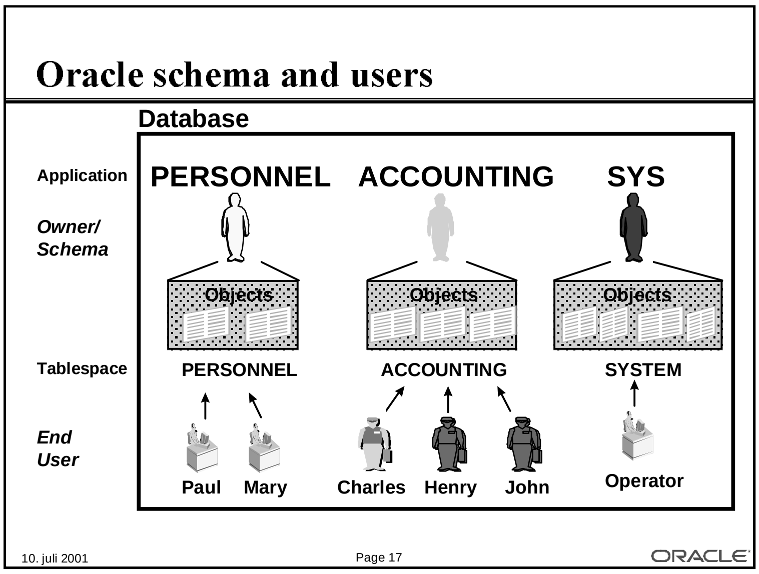


Bitmapped indexes

```
  <Blue, 1000100100010010100>
 <Green, 0001010000100100000>
   <Red, 0100000011000001001>
<Yellow, 0010001000001000010>
```

10. juli 2001 Page 16 ORACLE

In a *bitmapped index*, the index tree contains (compressed) bitmaps rather than rowid’s. Each bitmap lists all rows that have the indexed column identical to the index value. Bitmapped indexes are good for low-cardinality columns, i.e. columns where only few different values exist.



Oracle schema and users

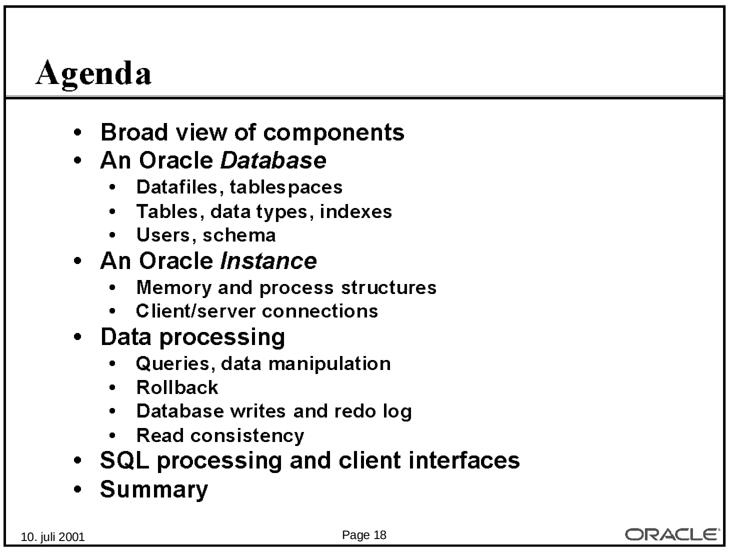
10. juli 2001 Page 17 ORACLE

Since there is typically only one Oracle database on each server, Oracle must contains means to cotrol access at a level different from typical operating system level. Oracle has the concept of *users*, who are real persons allowed to access data in the database and *schema*, which are pseudo-users actually allowed to create objects such as tables or indexes.

A database *user*, is an identified person, who can log in to the database. Each database user has a username and a password, and a special user is the operator who can do system administration.

All objects (tables, indexes, stored procedes, etc) of the database belong to a specific *schema*. Typically (but not necessarily) each application using the database has a schema (and one or more tablespaces) associated with it, and access rights are granted by the application owner schema to the actual users of that application.

Note, that in the current version of Oracle9i (9.0.1) a user and a schema are identical, but it will be different to clearly distinguish application owners and actual users in a later release.

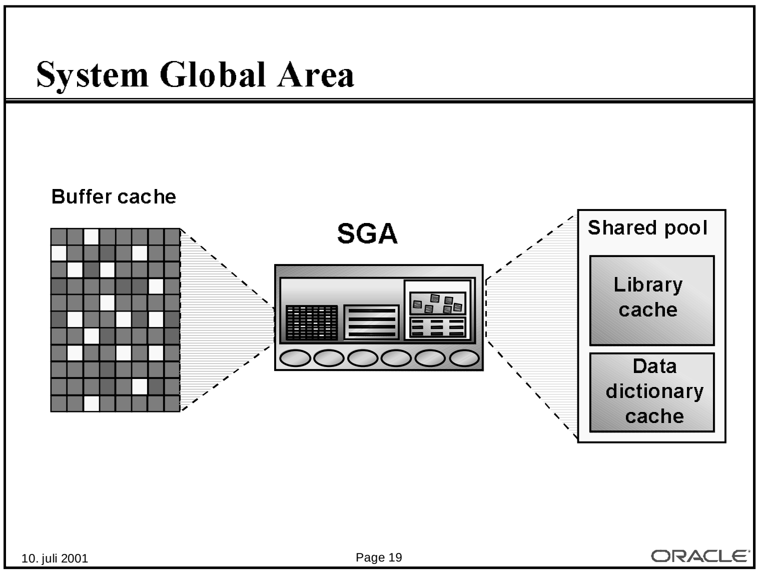


Agenda

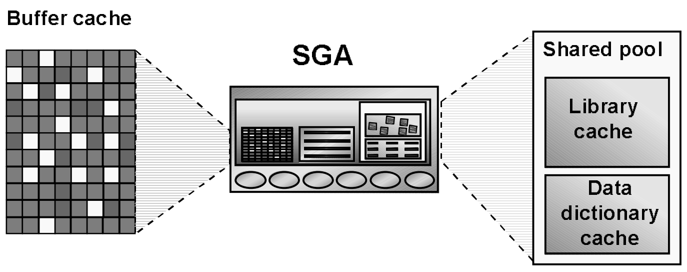
- Broad view of components
- An Oracle *Database*
  - Datafiles, tablespaces
  - Tables, data types, indexes
  - Users, schema
- An Oracle *Instance*
  - Memory and process structures
  - Client/server connections
- Data processing
  - Queries, data manipulation
  - Rollback
  - Database writes and redo log
  - Read consistency
- SQL processing and client interfaces
- Summary

10. juli 2001 Page 18 ORACLE

The Oracle *Instance* is a set of processes and shared memory. The instance is can be running, in which case Oracle is available, or stopped, in which case Oracle is unavailable. When the instance is running, certain data structures in shared memory and certain processes are seen.



System Global Area



10. juli 2001 Page 19

The large shared memory segment, the SGA, contains the following major components:

• The buffer cache, which is a cache of disk blocks, very similar to the file system cache found in most operating systems. Blocks are always read and written in sizes of the Oracle Block Size, which is defined, when the database is created.

• The shared pool, which contains two main components:

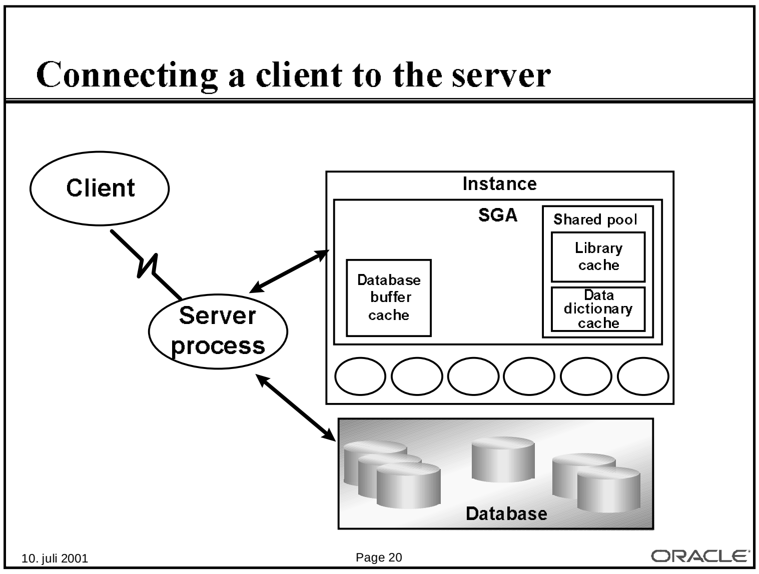
  • The *library cache*, which is a cache of SQL statements, etc.

  • The *dictionary cache,* which caches Oracle own internal information, such as information about users and tables.

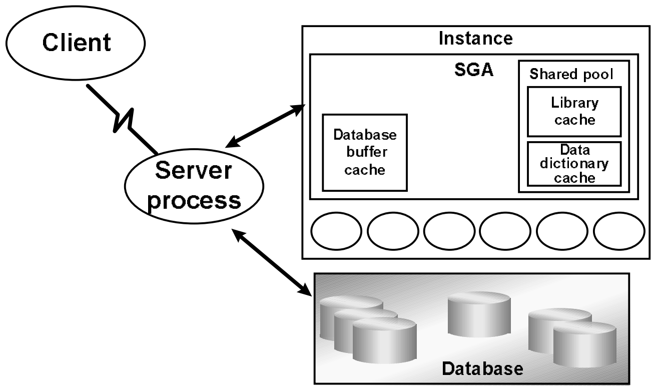
• As a very rough rule of thumb, the buffer cache and the shared pool, which by far are the largest of the SGA, will each make up little less than half the total SGA size.

• Other parts of the SGA contain information about currently running processes, locks, etc.

The use of some of the background processes will be explained later.



Connecting a client to the server

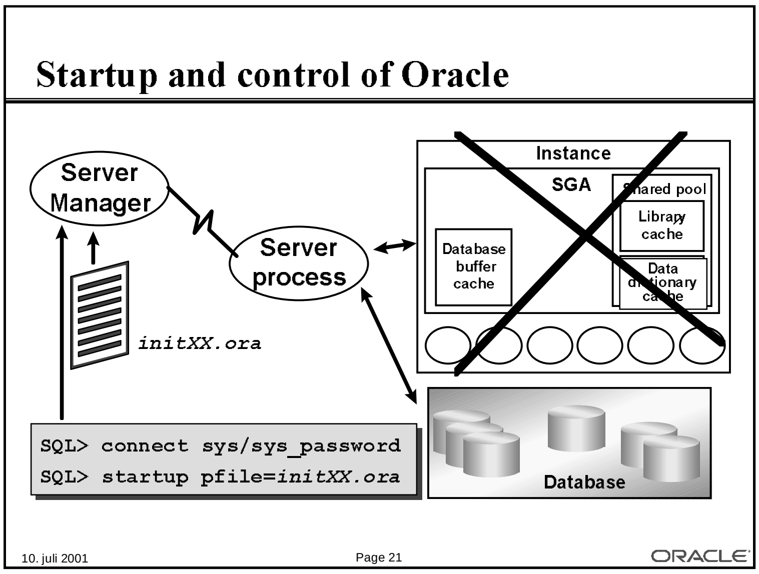


10. juli 2001 Page 20 ORACLE

• The *client* process runs in its own address space (actually as its own operating system process, quite often on a separate computer), completely separated from the Oracle Instance by the network or the inter process communication. Hence, the client cannot attach to the SGA, and cannot read or write to the database files.

• The *server* process, which executes SQL statements sent from the client process, executes within the address space of the SGA, and within the priveledge space of the database, and can hence get and modify information in the SGA, and can read and write to the database files.

• Full security is guaranteed by the Oracle security mechnisms and the SQL language.



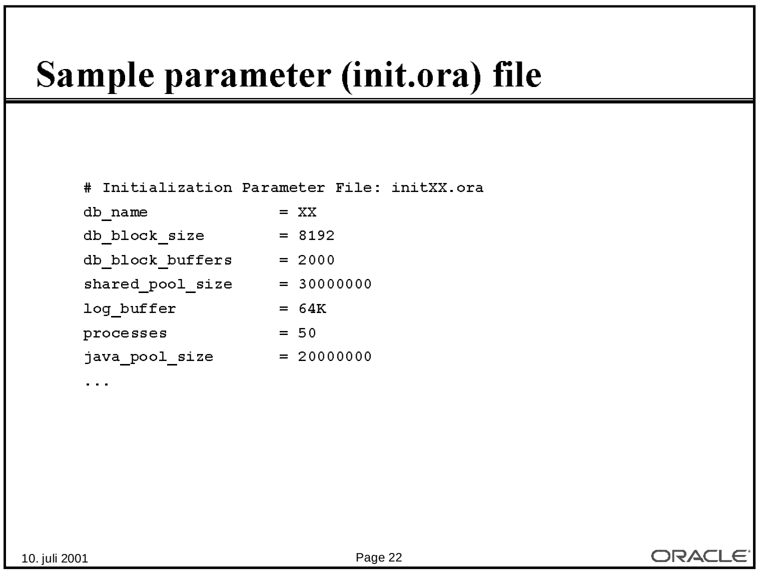
Startup and control of Oracle

```
SQL> connect sys/sys_password
SQL> startup pfile=initXX.ora
```

10. juli 2001 Page 21

Right after system startup, only the Oracle *database* will exist; the *instance* will not be running. To start the instance, a user with DBA authority connects to a server process using either the *Server Manager* utility or a GUI tool such as Oracle Enterprise Manager. The utility will read a parameter file, which has information about different things such size of the buffer cache and the shared pool, and about the maximum number of processes to start. The server process then creates the SGA and starts the necessary background processes.

On Unix systems, this is typically done in a system startup script, such as /etc/rc, and on Windows NT, this is done as a service.

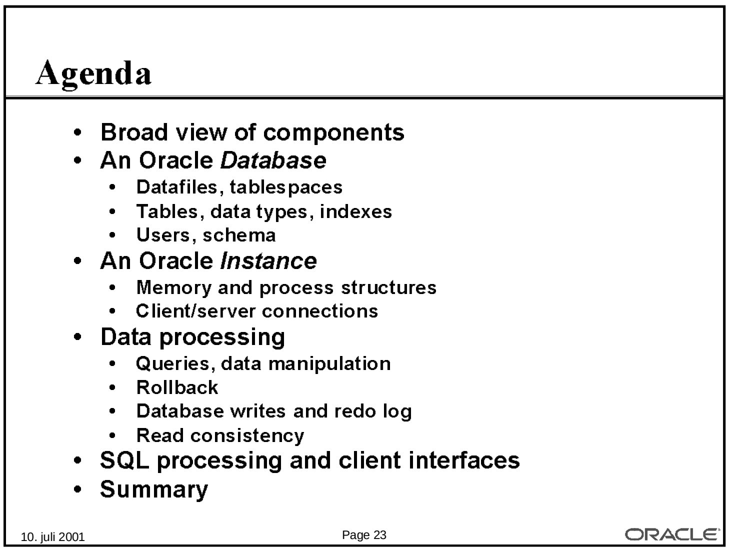


Sample parameter (init.ora) file

```
# Initialization Parameter File: initXX.ora
db_name              = XX
db_block_size        = 8192
db_block_buffers     = 2000
shared_pool_size     = 30000000
log_buffer           = 64K
processes            = 50
java_pool_size       = 20000000
...
```

10. juli 2001 Page 22

During startup of the Oracle instance, a parameter file is read, specifying information on how to configure Oracle. Some, but not all, of the parameters can later be modified at runtime either at the system level for the complete instance, or at the session level for a specific session.

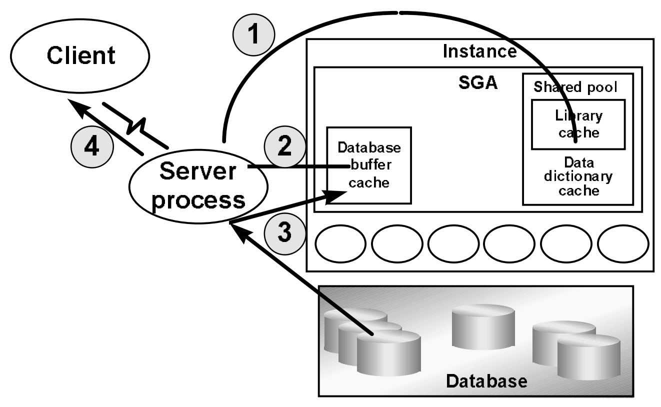


Agenda

- Broad view of components
- An Oracle *Database*
  - Datafiles, tablespaces
  - Tables, data types, indexes
  - Users, schema
- An Oracle *Instance*
  - Memory and process structures
  - Client/server connections
- Data processing
  - Queries, data manipulation
  - Rollback
  - Database writes and redo log
  - Read consistency
- SQL processing and client interfaces
- Summary

10. juli 2001 Page 23 ORACLE

Dataflow during data retrieval



10. juli 2001 Page 24

During queries (SQL select statements) data is found in the actual database, i.e. in the database files, and the server process, that is connected to the client together with the other parts of the Oracle instance is responsible for executing the SQL statement, getting data from the database and sending these to the client.

The major steps in this are:

1. The SQL statement is looked up in the *library cache* part of the SGA. If it is not already there, it will have to be parsed first.

2. The blocks containing the data to be retrieved, including e.g. index blocks, are identified in the *buffer cache* part of the SGA.

3. If the blocks do not exist, free buffers are found in the buffer cache, and the necessary blocks are read from the database files.

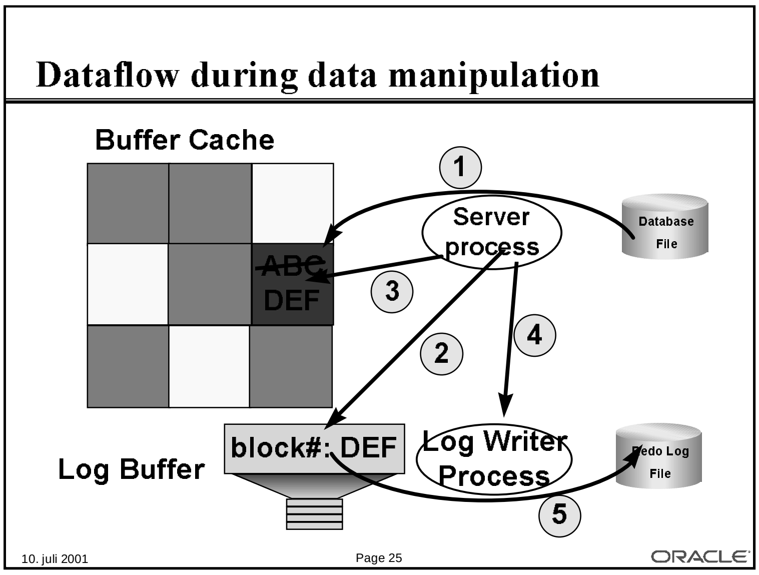
4. The information in the blocks are decoded into rows and columns, which are sent to the client.

There are some important things, that should be noted:

• Oracle uses caches at different places to ensure good performance of repetitive things. For the buffer cache, this is very much like an operating system file buffer cache; the behaviour and use of the library cache will become clear later.

• The actual read of data from the database files into the buffer cache is done by the server process. We shall see later, that writing of data is done by one of the background processes.

• The physical (on disk) storage of data is not seen by the client, which simply receives rows with columns of data as expected.



During processing of a data manipulation statement (DML, i.e. SQL insert, update or delete), the basic processing is like for queries. I.e. handling of SQL and the library cache is the same.

A new element is now added to the SGA: the *log buffer*. The log buffer logs all changes made to the buffer cache, so that these changes can be redone in case of a recovery. The processing steps are:

1. The block, that needs to me modified is read from the disk (unless, it is already found in the cache)

2. The server process (on behalf of the SQL statement executed by the client), makes an entry into the log buffer, specifying the operation to be done on the data block. The picture shows, that the redo log entry indicates which block is being updated (it is actually more than simply a block number), and it shows the actual change. No information is logged about the previous value in the block.

3. The server process makes the actual change in the block in the buffer cache. In the picture, the value “ABC” is replaced by the value “DEF”. At this point in time, the block in the buffer cache is different from the block in the datafile; this is called a dirty block.

4. The user issues the *commit* operation, and the server process indicates this to the *log writer* background process. This process is part of the Oracle instance, and is started when the instance starts.

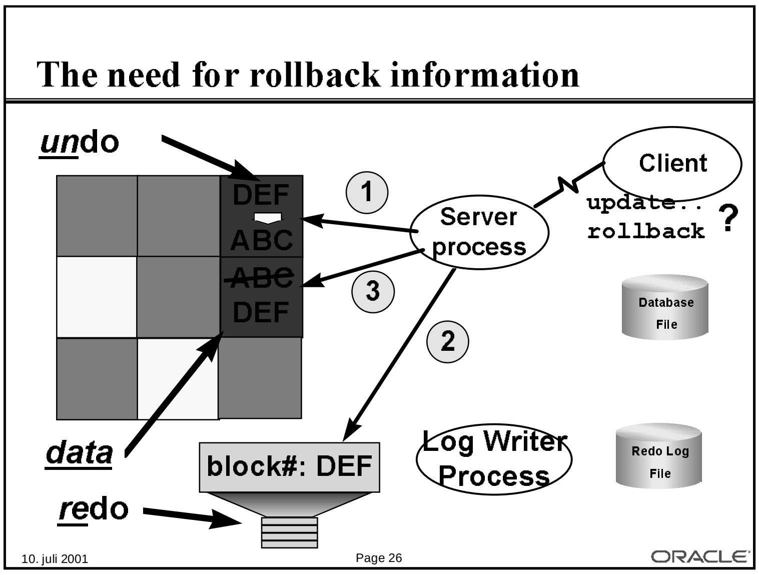
5. The log writer process writes the log buffer to a *redo log file*. At this time, in case of recovery, theold block, found in the database file, and the redo log record, found in the redo log file, can be used to redo the change made by the user.

New conecpts introduced:

• The *redo log buffer,* which is found in the SGA is used to store log information between an actual update and the coresponding commit. The typical size is up to a few hundred kilobytes.

• The *log writer* background process, often abbreviated to *LGWR,* which writes the redo log buffer to the redo log files.

• The *redo log* files, which is where Oracle stores log records necessary for recovery. Typical sizes are up to a few hundred megabytes or in vary high-transaction systems, a few gigabytes.



The need for rollback information

The previous slide showed the processing during normal DML operations, i.e. where the client eventually commits the operation. However, clients may want to undo the operation by performing a *rollback* in stead of the commit, and the simple picture does not show this. On this slide, the details of undo processing is shown.

1. Before doing the actual modification of the data block, information necessary to undo the operation is written into an *undo* block, which is also found in the buffer cache.

2. The redo log information is written to the redo log buffer.

3. The actual data block is modified. In order to reconstruct the data block to its state before the modification, the information in the rollback block is applied to the modfied block.

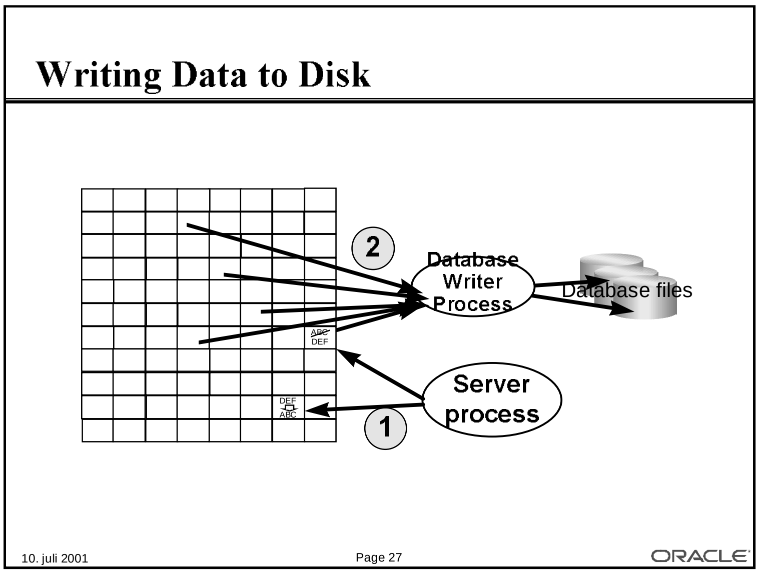
Important points

• Note the duality of the two words "redo" and "undo". Rollback, which is the normally used word, is synonymous with undo, which is frequently used in Oracle internal documents.

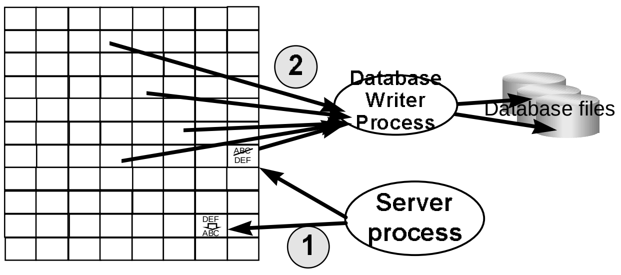
• The undo blocks are actually part of an *undo tablespace* or of *rollback segments(s)*, which are created and managed by the database administrator.

• The rollback segments are system controlled segments; they are stored in database files just like ordinary table and index segments, and the blocks of the rollback segment are cached in the buffer cache of the SGA. In Oracle9i, individual rollback segments are no longer used, rather a undo tablespace is created by the database administrator, and Oracle automatically manages space in there.

• The modification of the undo block (step 1 above) is actually logged in the redo log buffer, just as the modifcation of the actual data block.



Writing Data to Disk

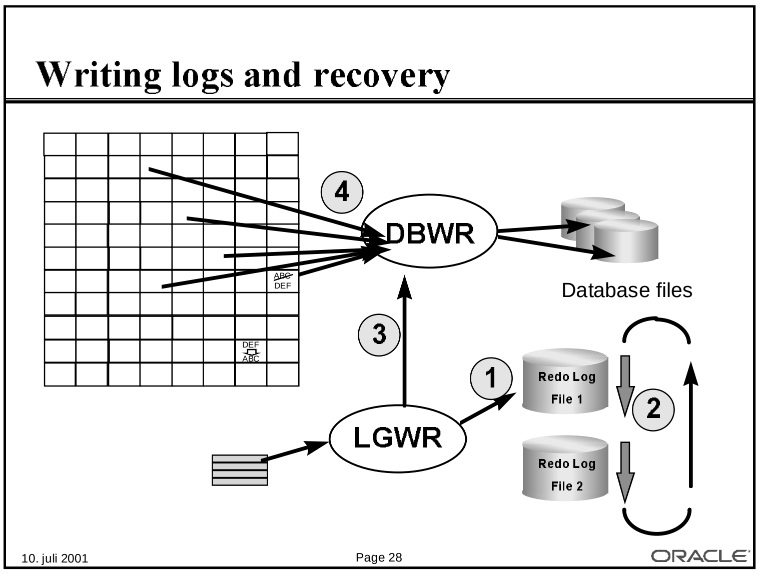


10. juli 2001 Page 27 ORACLE

As it has been seen earlier, whenever data is changed by a server process on behalf of a client's SQL request, the server process makes modifications to the blocks in the buffer cache, including rollback blocks, and writes redo information to the redo log buffer, which is written to disk by the log writer process at commit time. This implies that the buffer cache over time will be filled with modified blocks, so called dirty blocks. These blocks really need to be written to the database files. This is the mechanism for doing this:

1. Blocks are modfied in the buffer cache, and marked as dirty.

2. At regular intervals, or if no free buffers are found, the *database writer* process, abbreviated *DBWR*, will write dirty blocks to disk. For performance reasons, writing is done in batches and it will use asynchronous I/O.



Writing logs and recovery

This slide shows some details of the way the log writer and the database writer work together. In general, there are at least two redo log files, and the set of files are used sequentially; when one fills up, the next log buffer is written to the next redo log file.

1. The LGWR process writes log buffers to one of the redo log files. The dirty buffers in the buffer cache are not yet written to the database file, but in case of a recovery, the old blocks in the database file plus the information in the redo log file can be used to reconstruct the modified blocks.

2. The redo log file fills up, and should now be made to ready for the next cycle of redo log writing. This means, that before the current redo log files is being overwritten, all dirty buffers in the buffer cache must be flushed, so that recovery of the blocks are not necessary.

3. The LGWR instructs the DBWR, that a flush is necessay, this is called a *checkpoint*. The checkpoint must complete before the current redo logfile is being overwritten.

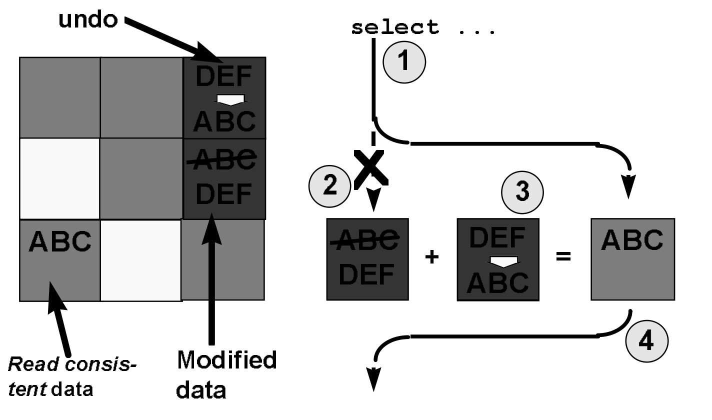
4. The DBWR executes the checkpoint, which means that all dirty buffers are being written to the database file.

Important points

• A *checkpoint*, which is a complete flush of all dirty buffers in the buffer cache, occurs at least whenever a redo log file becomes full. The database administrator my choose to make this happen more often.

• If a checkpoint does not finish before the complete cycle of redo log files, all operations in the Oracle Server will have to wait for the checkpoint to complete.

Read Consistency



10. juli 2001 Page 29 ORACLE

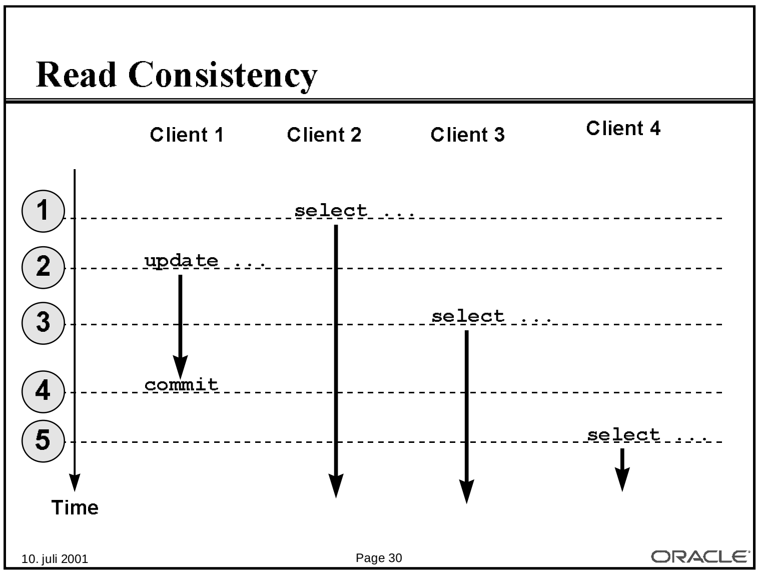
The rollback (undo) information, that has been discussed earlier, is used in other cases, besides an actual rollback operation from the client. In the scenario above, some user has updated the block with “ABC” into “DEF”, which is actually changed in the data block and registered in the undo block as rollback information. Concurrently, some other user is performing a query.

1. A user is performing a query, which reads blocks in the buffer cache.

2. A block is read, which is too new, i.e. it is modified after the start of the query. Hence, to get a consistent read for the query, this block cannot be used.

3. The information in the undo-block is applied to a copy of the modified block, with the effect that a block with the original block contents is constructed.

4. The query uses this *consistent read* copy of the block and continues.



The behaviour of the *read consistency* model is described further in this slide:

1. A user, client-2 is starting a query at time-1. The read consistency model ensures a consistent view of data during the entire run of the query, which is a snapshot of the database taken at the time the query starts.

2. The user, client-1 executes an update operation (which still may be either committed or rolled back).

3. The user, client-3, starts a query. Since the start of the query is before the commit of client-1's update operation, client-3 does not see the update done by client-1.

4. At this time, client-1 decides to do an update. However, since both queries of client-2 and client-3 started earlier than the commit-time, none of the queries will see the update.

5. Client-4 starts a query after the commit of client-1, hence the query started at time-5 does see the change made by client-1.

Important points:

• The snapshot time of a query is the start time of the query, independent on the time taken to complete the query.

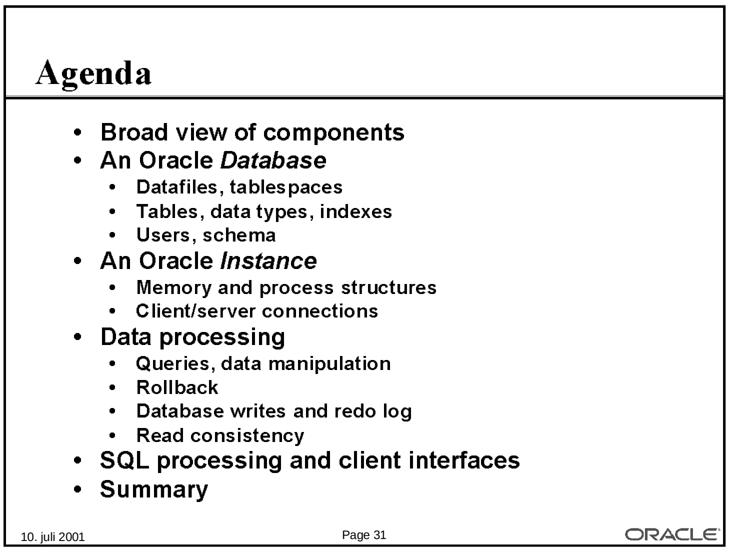
• Effective timestamp of DML operations (insert, update, delete) is the commit time, and only the client actually performing the DML operation, is able to see the changes made until commit time.

• The rollback segments are used by all other clients to reconstruct a read consistent view of data, taken at the start time of the query.

• Due to the fact, that rollback segments may be needed by queries after the commit of an operation, rollback information is not released immediately after commit.

• Rollback segments are in fact circurar buffers, which means that rollback information is being overwritten after a certain period of time. With Oracle9i, this is managed automatically, by the database administrator specfying the time to keep undo information available.

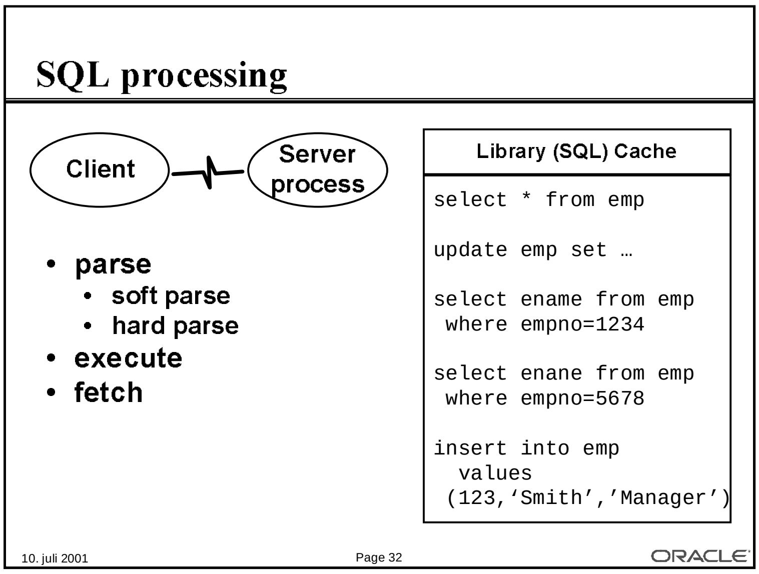
• If very long running queries (several minutes to hours) are executing at the same time as there is a risk ageing out necessary rollback information for the query. In this case, an error will be returned to the client. Note, however, that this is far better than using other database systems, where concurrently running updates and queries are either impossible due to locking, or errorneous due to queries returning in-consistent results.



Agenda

- Broad view of components
- An Oracle *Database*
  - Datafiles, tablespaces
  - Tables, data types, indexes
  - Users, schema
- An Oracle *Instance*
  - Memory and process structures
  - Client/server connections
- Data processing
  - Queries, data manipulation
  - Rollback
  - Database writes and redo log
  - Read consistency
- SQL processing and client interfaces
- Summary

10. juli 2001 Page 31 ORACLE



SQL processing

Client
Server process

- **parse**
  - **soft parse**
  - **hard parse**
- **execute**
- **fetch**

Library (SQL) Cache

```
select * from emp

update emp set …

select ename from emp
 where empno=1234

select enane from emp
 where empno=5678

insert into emp
  values
 (123,‘Smith’,’Manager’)
```

10. juli 2001 Page 32 ORACLE

Oracle SQL processing is done in generally three steps via the client/server interface; SQL statements are sent to the server, are being executed as the server and results are sent back to the client. For optimization, the number of roundtrips between the server and the client should be limited, as each roundtrip incurs a network message or a process switch if the client and the server runs on the same computer.

The **parse** step

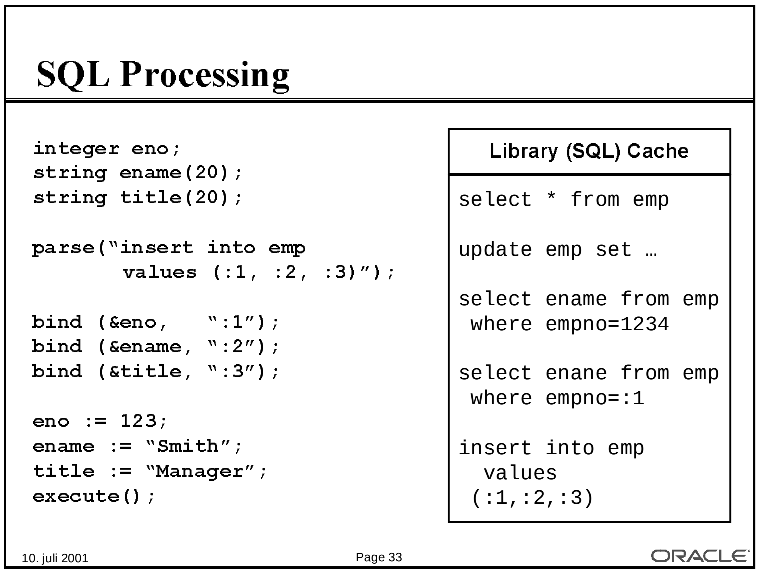
In order for a SQL statement to be executed, is must be found in the Library Cache of the SGA, and must be prepared for exeuction. This is called the *parse* step. The SQL statement is sent to the server, which first tries to identify an identical statement in the cache; if one is found, a *soft parse* is executed, which mainly does a verification of access rights. If the SQL statement is not found, a *hard parse* is done, which includes actual parsing and optimization of the statement. The hard parse can be time consuming, in particular for complex SQL statement, but even relatively simple SQL statements have a parse overhead, that can (and should) be avoided.

The **execute** step

After parsing, the client instructs the server to execute the SQL statement. This step can be repeated, which means that a SQL statement can be parsed once and executed several times, which may heavily reduce the overhead by parsing.

The **fetch** step

If the SQL statement is a query, the rows that are the result are sent to the client during this step.



SQL Processing

```
integer eno;
string ename(20);
string title(20);

parse("insert into emp
       values (:1, :2, :3)");

bind (&eno,   ":1");
bind (&ename, ":2");
bind (&title, ":3");

eno := 123;
ename := "Smith";
title := "Manager";
execute();
```

Library (SQL) Cache

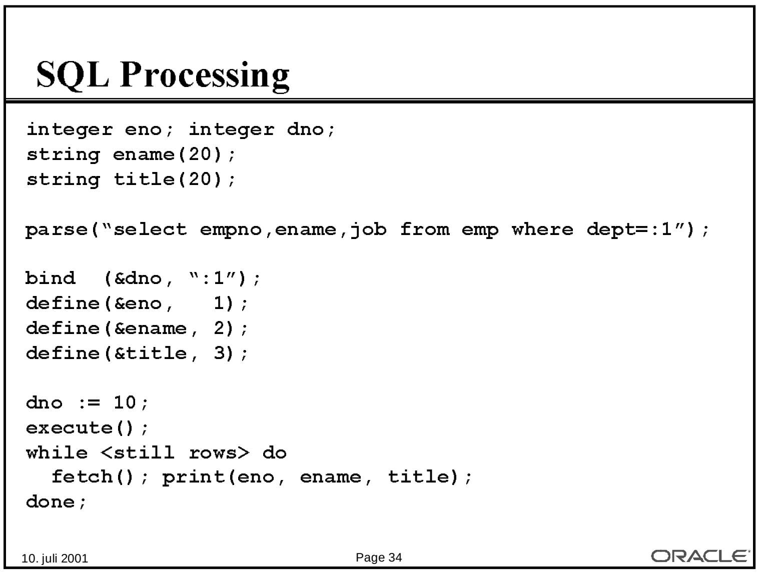
```
select * from emp

update emp set …

select ename from emp
 where empno=1234

select enane from emp
 where empno=:1

insert into emp
  values
 (:1,:2,:3)
```

10. juli 2001 Page 33 ORACLE

In order to reduce the number of parse operations, all SQL statements that are being used repetitively should use *placeholders*, also called *bind-variables*, as the ":1" in the select statement in the program example above. The example shows an extra processing step:

The **bind** step

Any constant in SQL statements, e.g. the '1234' in 'select * from emp where empno=1234' can, and generally should, be replaced by *bind*-variables, which are identified by colon followed by a number or a string, e.g. :1, :2, :abc, etc. In the client program, an actual variable is *bound* to this bind-variable, and the actual value of the program variable is then used at execute time. This means that the execute can be repeated with different values, but with the same SQL statement; hence, no new parse is necessary.



SQL Processing

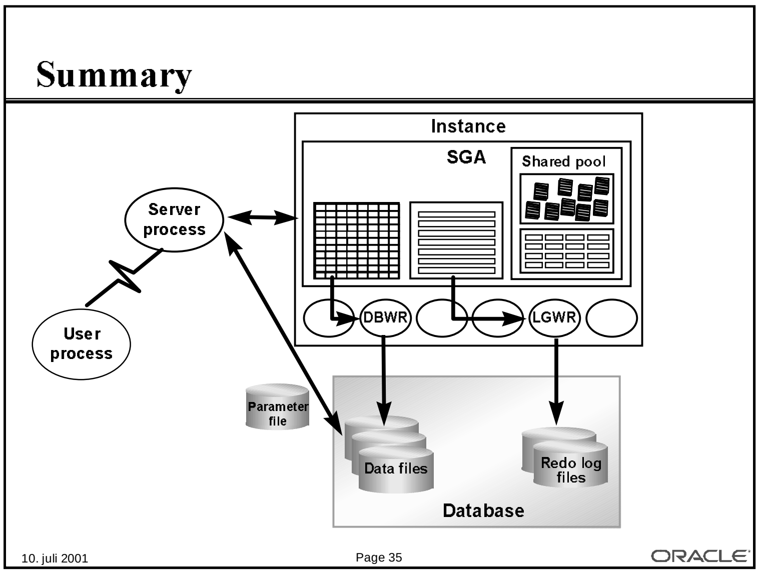
```
integer eno; integer dno;
string ename(20);
string title(20);

parse("select empno,ename,job from emp where dept=:1");

bind  (&dno, ":1");
define(&eno,   1);
define(&ename, 2);
define(&title, 3);

dno := 10;
execute();
while <still rows> do
  fetch(); print(eno, ename, title);
done;
```

10. juli 2001 Page 34 ORACLE

Bind-variable can (and should) also be used with queries as in this example. Additionally, this example shows the *define* step:

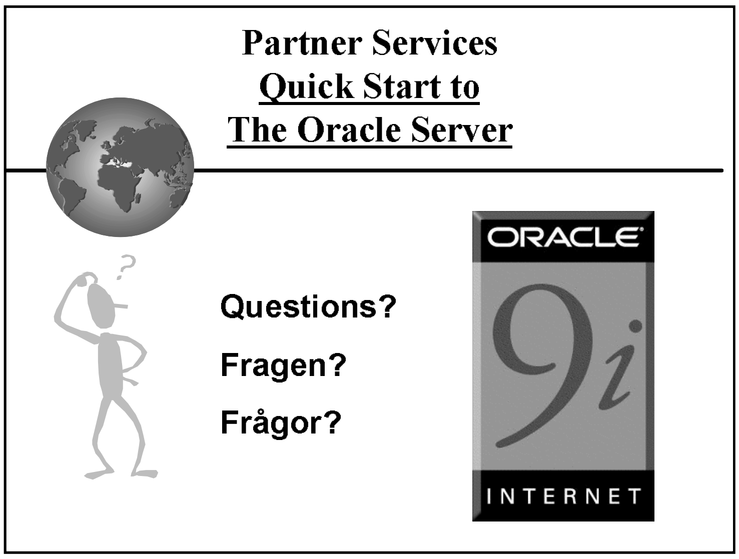
The **define** step

For queries, program variables must be available to retrieve the results of the query during the fetch step. This is done by the *define* call, which takes program variable and the number in the select-list as arguments.



Summary

10. juli 2001 Page 35 ORACLE



Partner Services
Quick Start to The Oracle Server

Questions?

Fragen?

Frågor?

ORACLE 9i INTERNET