Implementation of Database Systems

236510

David Konopnicki

Taub 715

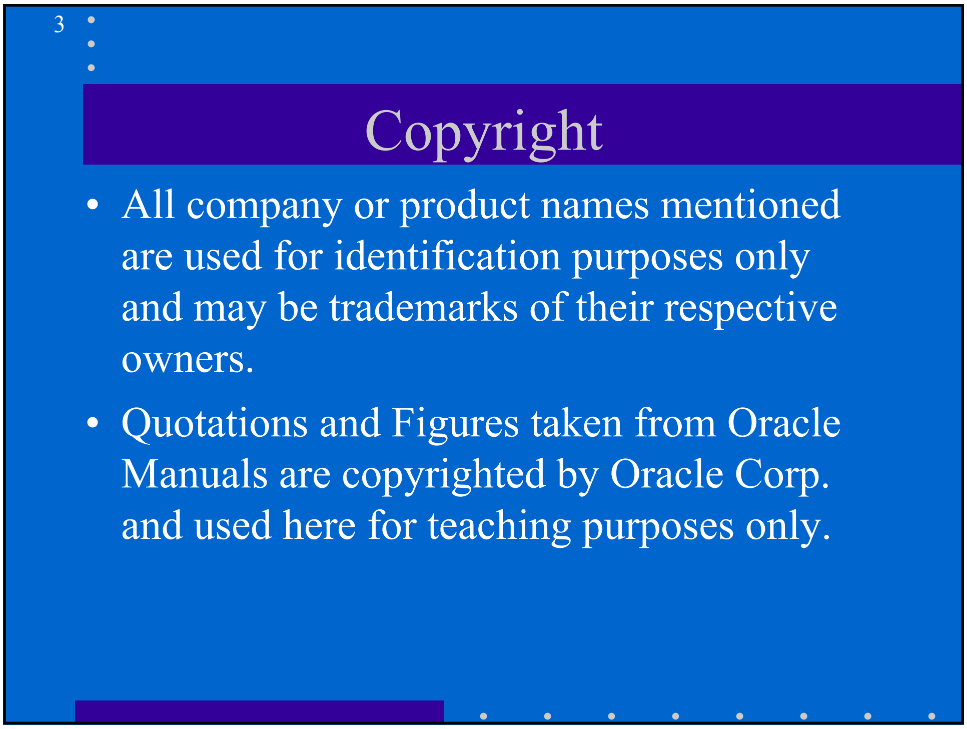
Spring 2000

Sources

- Oracle 7 Server Concepts - Oracle8i Server Concepts. © Oracle Corp.
- Available on the course Web Site:

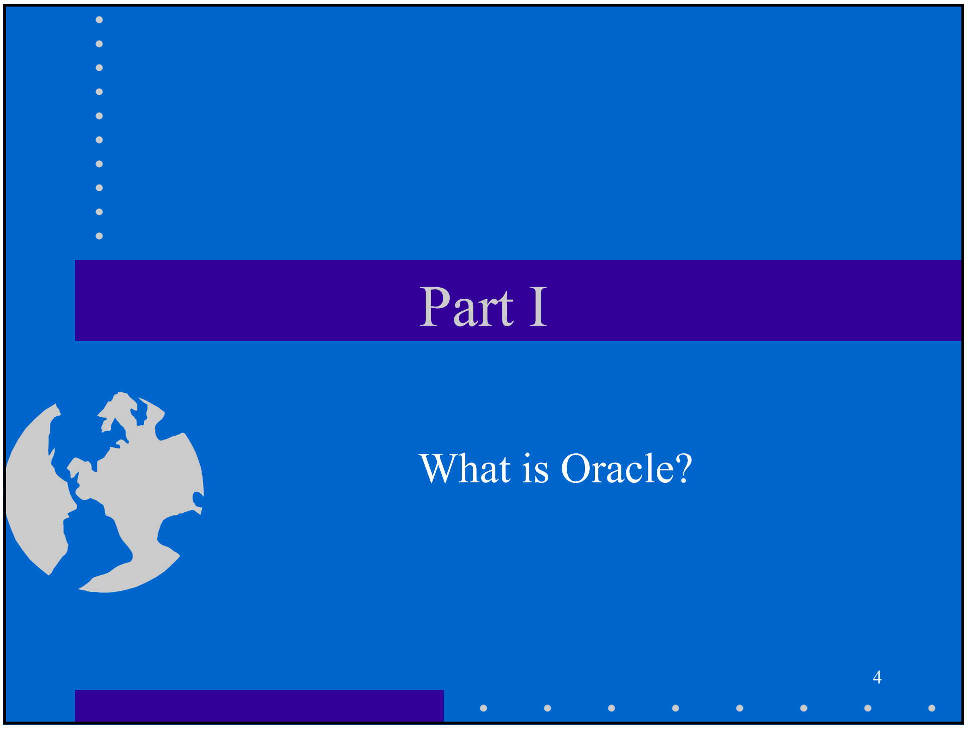
http://www.cs.technion.ac.il/~cs236510

- We will learn the internal structure of Oracle7, a modern RDBMS.



Copyright

- All company or product names mentioned are used for identification purposes only and may be trademarks of their respective owners.
- Quotations and Figures taken from Oracle Manuals are copyrighted by Oracle Corp. and used here for teaching purposes only.



Part I

What is Oracle?

What is a Database

A computer system that manages information. In general, a database server must reliably manage a *large amount* of data in a *multi-user* environment so that many users can *concurrently* access the same data. All this must be accomplished while delivering *high performance*. A database server must also *prevent unauthorized access* and provide efficient solutions for *failure recovery*.

Oracle Features

- Client/server architecture
- Large DB: up to 1 terabyte (2^{40} bytes)
- Many concurrent database users
- High performance
- High availability
- Controlled availability
- Manageable security
- Data integrity
- Distributed systems
- Replication

The Oracle Server

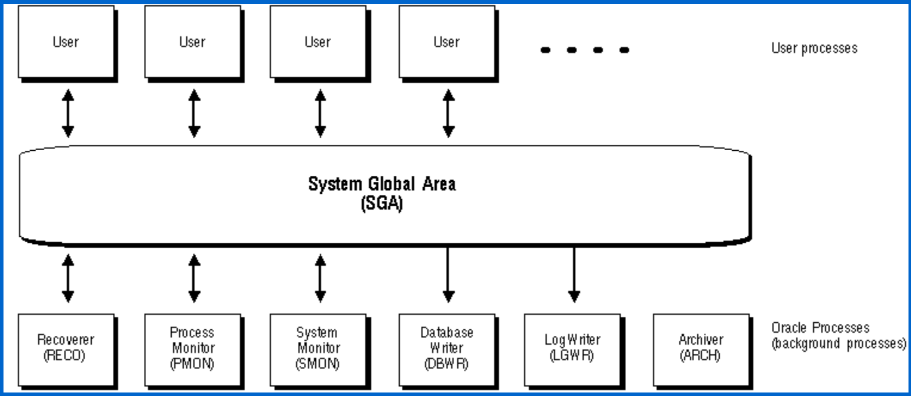
- The Oracle server consists of an Oracle database and an Oracle instance.
- Database structure:
  - Physical DB struct: determined by the OS. Three types of files: datafiles, redo log files, control files.
  - Logical DB struct: Tablespaces, Schema objects.

The Oracle Server (2)

An Oracle Instance:

- Every time a database is started, a system global area (SGA) is allocated and Oracle background processes are started.
- The system global area is a an area of memory used for database information shared by the database users.
- The combination of the background processes and memory buffers is called an Oracle *instance.*

The Oracle Server (3)



The Oracle Server (4)

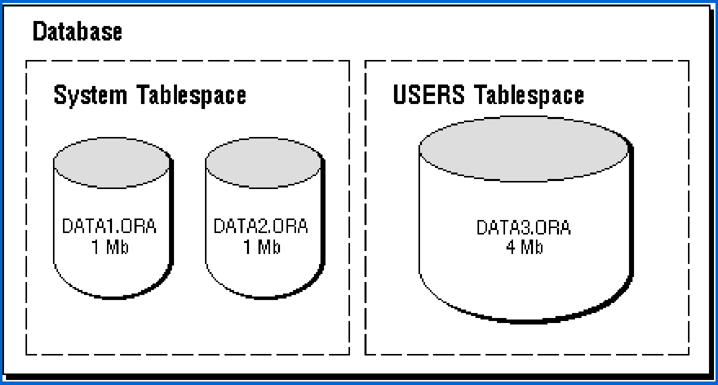
- If the user and server processes are on different computers of a network, the user process and server process communicate using SQL*Net. *SQL*Net* is Oracle's interface to standard communications protocols that allows for the proper transmission of data between computers.

Database Structure

- *Structures*: Structures are well-defined objects (such as tables, views, indexes, and so on) that store or access the data of a database. Structures and the data contained within them can be manipulated by operations.
- *Operations*: Operations are clearly defined actions that allow users to manipulate the data and structures of a database. The operations on a database must adhere to a predefined set of integrity rules.
- *Integrity Rules*:Integrity rules are the laws that govern which operations are allowed on the data and structures of a database. Integrity rules protect the data and the structures of a database.

Logical Database Structures

- Databases, tablespaces, datafiles



Logical Database Structures (2)

- Schema objects: **Tables**
  - A *table* is the basic unit of data storage in an Oracle database.
  - The tables of a database hold all of the user-accessible data.
  - Table data is stored in *rows* and *columns*. Every table is defined with a *table name* and set of columns.
  - Each column is given a *column name*, a *datatype* and a *width.*
  - Once a table is created, valid rows of data can be inserted into it. The table's rows can then be queried, deleted, or updated.
  - To enforce defined business rules on a table's data, integrity constraints and triggers can also be defined for a table.

Logical Database Structures (3)

- Schema Objects: Views
  - A *view* is a custom-tailored presentation of the data in one or more tables. A view can also be thought of as a "stored query".
  - Like tables, views can be queried, updated, inserted into, and deleted from, with restrictions. All operations performed on a view actually affect the base tables of the view.
  - Views are often used to do the following:
    - Provide an additional level of table security by restricting access to a predetermined set of rows and columns of a table.
    - Hide data complexity.
    - Simplify commands for the user.
    - Present the data in a different perspective from that of the base table.
    - Store complex queries.

Logical Database Structures (4)

- Schema Objects: Sequences
  - A *sequence* generates a serial list of unique numbers for numeric columns of a database's tables. Sequences simplify application programming by automatically generating unique numerical values for the rows of a single table or multiple tables.
  - Sequence numbers are independent of tables, so the same sequence can be used for one or more tables. After creation, a sequence can be accessed by various users to generate actual sequence numbers.

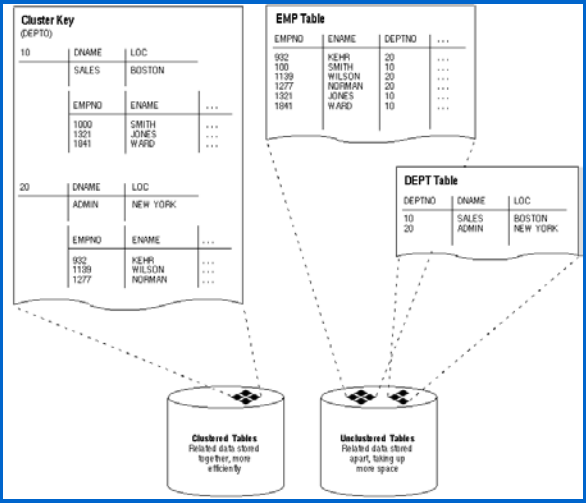
Logical Database Structures (5)

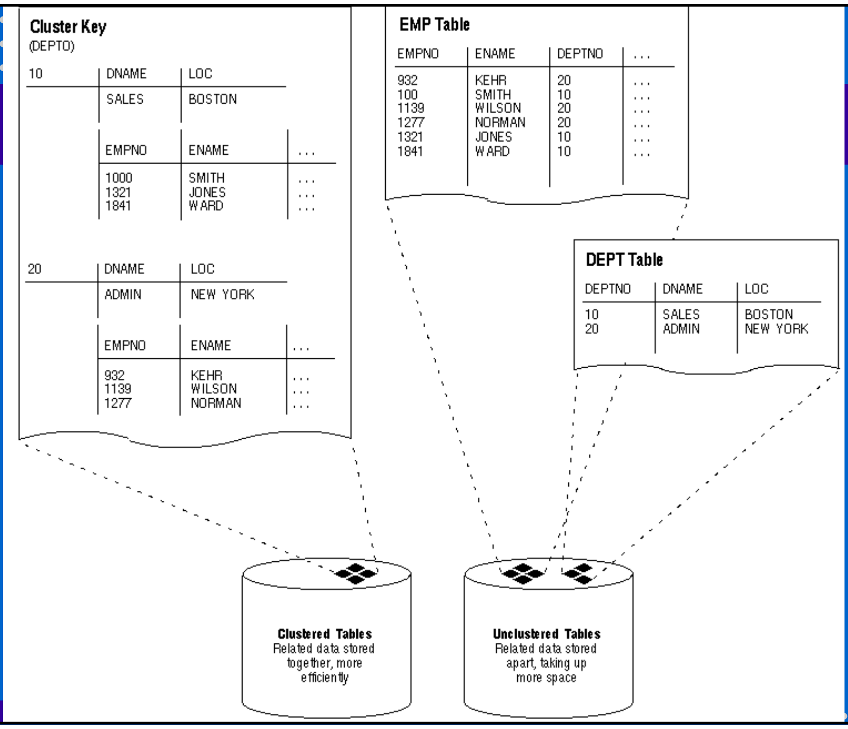
- Schema Object: Program Unit
  - The term "program unit" is used to refer to stored procedures, functions, and packages.
  - A *procedure* or *function* is a set of SQL and PL/SQL (Oracle's procedural language extension to SQL) statements grouped together as an executable unit to perform a specific task.
  - Stored procedures.

Logical Database Structures (6)

- Schema Objects: Indexes, Clusters, Hash Clusters
  - *Indexes* are created on one or more columns of a table. Once created, an index is automatically maintained and used by Oracle. Changes to table data are automatically incorporated into all relevant indexes with complete transparency to the users.
  - *Clusters* are groups of one or more tables physically stored together because they share common columns and are often used together.
  - *Hash clusters:* a row is stored in a hash cluster based on the result of applying a *hash function* to the row's cluster key value. All rows with the same hash key value are stored together on disk.

Logical Database Structures (7)


Cluster Key
(DEPTO)
10 | DNAME | LOC
SALES | BOSTON
EMPNO | ENAME | ...
1000 1321 1841 | SMITH JONES WARD
20 | DNAME | LOC
ADMIN | NEW YORK
EMPNO | ENAME | ...
932 1139 1277 | KEHR WILSON NORMAN
EMP Table
EMPNO | ENAME | DEPTNO | ...
932 100 1139 1277 1321 1841 | KEHR SMITH WILSON NORMAN JONES WARD | 20 10 20 20 10 10
DEPT Table
DEPTNO | DNAME | LOC
10 20 | SALES ADMIN | BOSTON NEW YORK
Clustered Tables
Related data stored together, more efficiently
Unclustered Tables
Related data stored apart, taking up more space


Cluster Key
(DEPTO)
EMP Table
DEPT Table
Clustered Tables
Related data stored together, more efficiently
Unclustered Tables
Related data stored apart, taking up more space

Cluster Key (DEPTO)

10	DNAME	LOC
	SALES	BOSTON

EMPNO	ENAME	. . .
1000	SMITH	. . .
1321	JONES	. . .
1841	WARD	. . .

20	DNAME	LOC
	ADMIN	NEW YORK

EMPNO	ENAME	. . .
932	KEHR	. . .
1139	WILSON	. . .
1277	NORMAN	. . .

EMP Table

EMPNO	ENAME	DEPTNO	. . .
932	KEHR	20	. . .
100	SMITH	10	. . .
1139	WILSON	20	. . .
1277	NORMAN	20	. . .
1321	JONES	10	. . .
1841	WARD	10	. . .

DEPT Table

DEPTNO	DNAME	LOC
10	SALES	BOSTON
20	ADMIN	NEW YORK

Logical Database Structures (8)

- Disk Space: Data Blocks
  - At the finest level of granularity, an Oracle database's data is stored in *data blocks*.
  - One data block corresponds to a specific number of bytes of physical database space on disk.
  - A data block size is specified for each Oracle database when the database is created.
  - A database uses and allocates free database space in Oracle data blocks.

Logical Database Structures (9)

- Disk Space: Extents
  - An *extent* is a specific number of contiguous data blocks, obtained in a single allocation, used to store a specific type of information.

Logical Database Structures (10)

- Disk Space: Segment
  - A *segment* is a set of extents allocated for a certain logical structure. the different types of segments include the following:
    - Data Segments: All of the table's data is stored in the extents of its data segment.
    - Index Segments:Each index has an index segment that stores all of its data.
    - Rollback Segments
    - Temporary Segments

Physical Database Structure (1)

- Datafiles:
  - Every Oracle database has one or more physical *datafiles*.
  - A database's datafiles contain all the database data (e.g. tables and indexes).
  - characteristics of datafiles:
    - Associated with only one database.
    - Automatically extend when the database runs out of space.
    - One or more datafiles form a logical unit of database storage called a tablespace.

Physical Database Structures (2)

- Using Datafiles:
  - The data in a datafile is read, as needed, during normal database operation and stored in the memory cache of Oracle.
  - Modified or new data is not necessarily written to a datafile immediately. To reduce the amount of disk access and increase performance, data is pooled in memory and written to the appropriate datafiles all at once, as determined by the DBWR background process of Oracle.

Physical Database Structures (3)

- Redo Log Files:
  - Every Oracle database has a set of two or more *redo log files*. This is called the *redo log*.
  - Records all changes made to data. Should a failure prevent modified data from being permanently written to the datafiles, the changes can be obtained from the redo log and work is never lost.
  - To protect against a failure involving the redo log itself, Oracle allows a *multiplexed redo log* so that two or more copies of the redo log can be maintained on different disks.

Physical Database Structures (4)

- Control Files:
  - A control file contains entries that specify the physical structure of the database.
    - database name
    - names and locations of a database's datafiles and redo log files
    - time stamp of database creation
  - Like the redo log, Oracle allows the control file to be multiplexed for protection of the control file.

The Data Dictionary

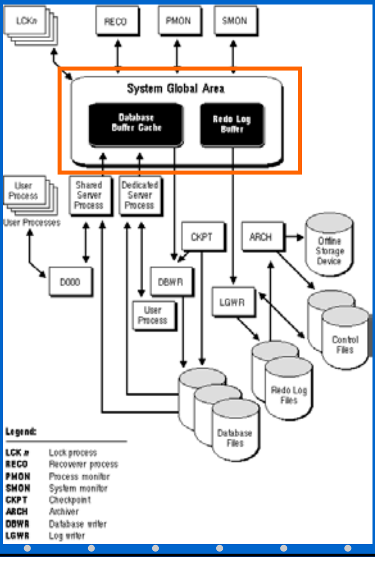
- An Oracle data dictionary is a set of tables and views that are used as a *read-only* reference about the database.
- Stores information about both the logical and physical structure of the database.
  - the valid users of an Oracle database
  - information about integrity constraints defined for tables in the database
  - how much space is allocated for a schema object and how much of it is being used

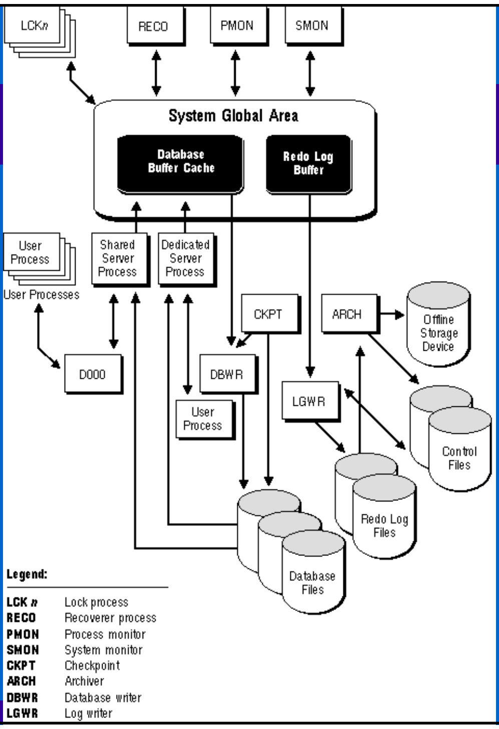
Server Architecture (1)

- An Oracle Server uses memory structures and processes to manage and access the database.
- All memory structures exist in the *main memory* of the computers that constitute the database system.
- *Processes* are jobs or tasks that work in the memory of these computers.

Server Architecture (2)

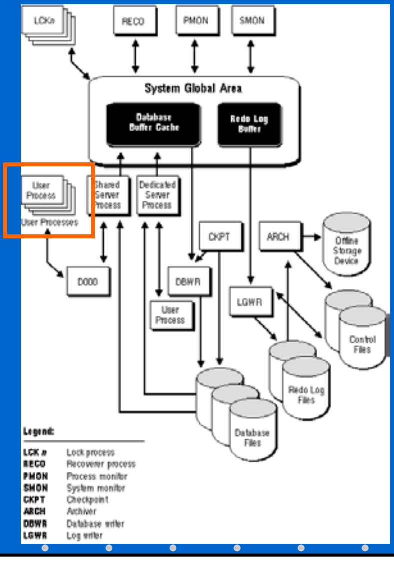
- Users currently connected to an Oracle Server share the data in the system global area.
- For optimal performance, the entire system global area should be as large as possible (while still fitting in real memory)
- Contain: the database buffers (for data), redo log buffer (for log entries), the shared pool (for SQL), and cursors.
- These areas have fixed sizes and are created during instance startup.





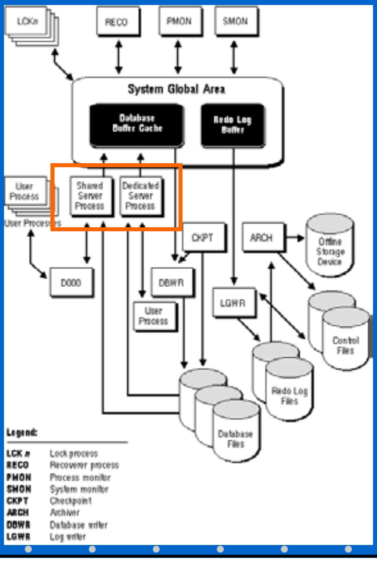
Server Architecture (3)

- A *user process* is created and maintained to execute the software code of an application program (such as a Pro*C/C++ program) or an Oracle tool (such as sqlplus). The user process also manages the communication with the server processes.



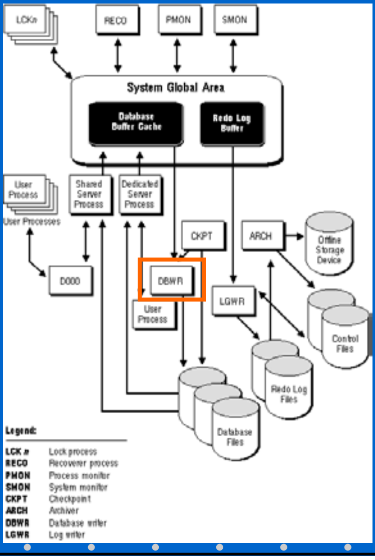
Server Architecture (4)

- Oracle creates *server processes* to handle requests from connected user processes.
- Carry out requests of the associated user process (e.g., read data into buffers).
- Oracle can be configured to vary the number of user processes per server process.



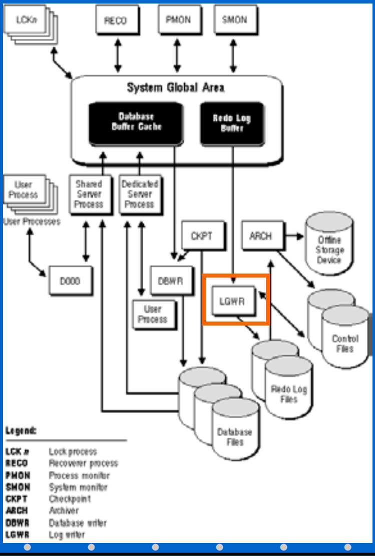
Server Architecture (5)

- Background Processes
- **Database Writer (DBWR)** writes modified blocks from the database buffer cache to the datafiles.
- DBWR does not need to write blocks when a transaction commits
- DBWR is optimized to minimize disk writes.



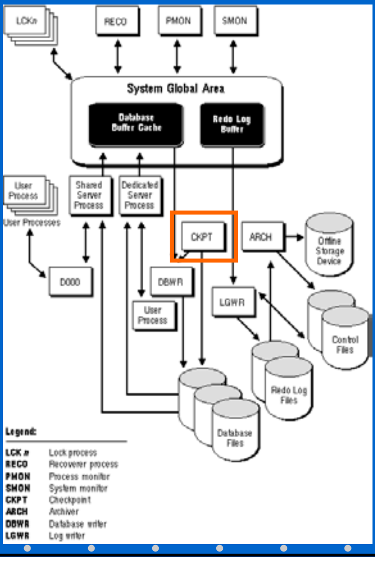
Server Architecture (6)

- Background Processes
- **Log Writer (LGWR)** The *Log Writer* writes redo log entries to disk.
- Redo log data is generated in the redo log buffer of the system global area.
- As transactions commit and the log buffer fills, LGWR writes redo log entries into an online redo log file.



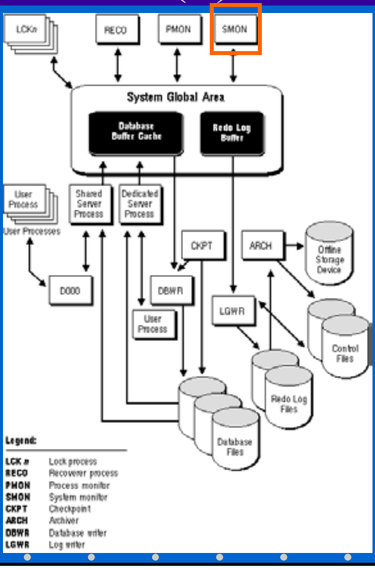
Server Architecture (7)

- Background Processes
- **Checkpoint (CKPT)** At specific times, all modified database buffers are written to the datafiles by DBWR; this is a checkpoint.
- CKPT is responsible for signaling DBWR at checkpoints and updating all the datafiles and control files to indicate the most recent checkpoint.



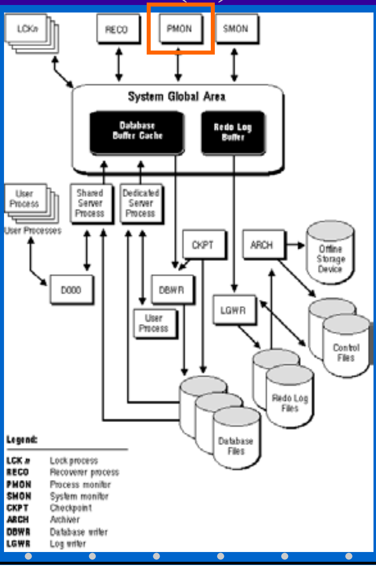
Server Architecture (8)

- Background Processes
- **System Monitor (SMON)** performs instance recovery at instance startup.
- SMON also coalesces free extents within the database to make free space contiguous and easier to allocate.



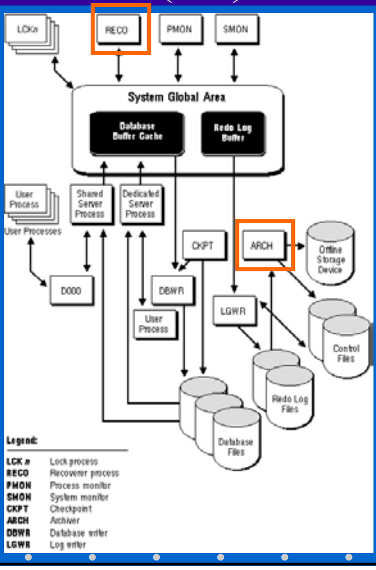
Server Architecture (9)

- **Process Monitor (PMON)** performs process recovery when a user process fails.
- PMON is responsible for cleaning up the cache and freeing resources that the process was using.
- PMON also checks on dispatcher and server processes and restarts them if they have failed.



Server Architecture (10)

- **Archiver (ARCH)** copies the online redo log files to archival storage when they are full.
- **Recoverer (RECO)** used to resolve distributed transactions pending due to a network or system failure in a distributed database.



Example of Work

- An instance is currently running on the computer that is executing Oracle.
- A computer used to run an application runs the application in a user process. The client application attempts to establish a connection to the server using the proper SQL*Net driver.
- The server is running the proper SQL*Net driver. The server detects the connection request from the application and creates a (dedicated) server process on behalf of the user process.
- The user executes a SQL statement and commits the transaction.
- The server process receives the statement and checks the shared pool for any shared SQL area that contains an identical SQL statement. If a shared SQL area is found, the previously existing shared SQL area is used to process the statement; if not, a new shared SQL area is allocated for the statement so that it can be parsed and processed.

Example of Work (2)

- The server process retrieves any necessary data values from the actual datafile (table) or those stored in the system global area.
- The server process modifies data in the system global area. The DBWR process writes modified blocks permanently to disk when doing so is efficient. Because the transaction committed, the LGWR process immediately records the transaction in the online redo log file.
- If the transaction is successful, the server process sends a message across the network to the application. If it is not successful, an appropriate error message is transmitted.
- Throughout this entire procedure, the other background processes run, watching for conditions that require intervention. In addition, the database server manages other users' transactions and prevents contention between transactions that request the same data.

Part II

Database Backup and Recovery

Introduction

- A major responsibility of the database administrator is to prepare for the possibility of hardware, software, network, process, or system failure.
- If such a failure affects the operation of a database system, you must usually recover the databases and return to normal operations as quickly as possible.
- Recovery should protect the databases and associated users from unnecessary problems and avoid or reduce the possibility of having to duplicate work manually.

Failures: User Error

- A database administrator can do little to prevent user errors (for example, accidentally dropping a table).
- Usually, user error can be reduced by increased training on database and application principles.
- Furthermore, the administrator can ease the work necessary to recover from many types of user errors (backups).

Failures: Statement Failure

- Statement failure occurs when there is a logical failure in the handling of a statement in an Oracle program (A valid INSERT statement cannot insert a row because there is no space available).
- If a statement failure occurs, the Oracle software or operating system returns an error code or message.
- A statement failure usually requires no action or recovery steps; Oracle automatically corrects for statement failure by rolling back the effects (if any) of the statement and returning control to the application. The user can simply re-execute the statement after correcting the problem conveyed by the error message.

Failures: Process Failures

- A failure in a user, server, or background process of a database instance
- When a process failure occurs, the failed subordinate process cannot continue work, although the other processes of the database instance can.
- The Oracle background process PMON detects aborted Oracle processes.
- If the aborted process is a user or server process, PMON resolves the failure by rolling back the current transaction.If the aborted process is a background process, you must shut down and restart the instance.

Failures: Network Failures

- A network failure might interrupt normal execution of a client application and cause a process failure to occur. In this case, the Oracle background process PMON detects and resolves the aborted server process for the disconnected user process, as described in the previous section.
- A network failure might interrupt the *two-phase commit* of a distributed transaction. Once the network problem is corrected, the Oracle background process RECO of each involved database server automatically resolves any distributed transactions not yet resolved at all nodes of the distributed database system.

Failures: DB Instance Failure

- Database instance failure occurs when a problem arises that prevents an Oracle database instance (SGA and background processes) from continuing to work. An instance failure can result from a hardware problem, such as a power outage, or a software problem, such as an operating system crash.
- Recovery from instance failure is relatively automatic. For example, in configurations that do not use the Oracle Parallel Server, the next instance startup automatically performs instance recovery. When using the Oracle Parallel Server, other instances perform instance recovery.

Failures: Disk Failures

- An error can arise when trying to write or read a file that is required to operate an Oracle database. This occurrence is called media failure because there is a physical problem reading or writing physical files needed for normal database operation.
- A common example of a media failure is a disk head crash, which causes the loss of all files on a disk drive. All files associated with a database are vulnerable to a disk crash, including datafiles, redo log files, and control files. The appropriate recovery from a media failure depends on the files affected

Failures: Disk Failures (2)

- Database operation after a media failure of online redo log files or control files depends on whether the online redo log or control file is *multiplexed*.
- If a media failure damages a single disk, and you have a multiplexed online redo log, the database can usually continue to operate without significant interruption.
- Damage to a non-multiplexed online redo log causes database operation to halt and may cause permanent loss of data.
- Damage to any control file, whether it is multiplexed or non-multiplexed, halts database operation once Oracle attempts to read or write the damaged control file.

Failures: Disk Failures (3)

- Media failures can be divided into two categories: read errors and write errors.
- In a read error, Oracle cannot read a datafile and an OS error is returned to the application, along with an Oracle error. Oracle continues to run, but the error is returned each time an unsuccessful read occurs. At the next checkpoint, a write error will occur when Oracle attempts to write the file header as part of the standard checkpoint process.
- If Oracle discovers that it cannot write to a datafile and Oracle archives filled online redo log files, Oracle returns an error in the DBWR trace file, and Oracle takes the datafile offline automatically. Only the datafile that cannot be written to is taken offline.

Failures: Disk Failures (4)

- If the datafile that cannot be written to is in the SYSTEM tablespace, the file is not taken offline. Instead, an error is returned and Oracle shuts down the database.
- If Oracle discovers that it cannot write to a datafile, and Oracle is not archiving filled online redo log files, DBWR fails and the current instance fails.
- If the problem is temporary, instance recovery usually can be performed using the online redo log files, in which case the instance can be restarted.
- If a datafile is permanently damaged and archiving is not used, the entire database must be restored using the most recent backup.

Recovery Structures: Backups

- A database backup consists of operating system backups of the physical files that constitute an Oracle database.
- To begin database recovery from a media failure, Oracle uses file backups to restore damaged datafiles or control files.

Recovery Structures: Redo Log

- Records all changes made in an Oracle database.
- The redo log of a database consists of at least two redo log files that are separate from the datafiles.
- As part of database recovery, Oracle applies the appropriate changes in the database's redo log to the datafiles, which updates database data to the instant that the failure occurred.
- A redo log can be comprised of two parts: the online redo log and the archived redo log.

Recovery Structures: Redo Log (2)

- Every Oracle database has an associated online redo log.
- The online redo log works with the Oracle background process LGWR to immediately record all changes made through the associated instance.
- The online redo log consists of two or more pre-allocated files that are reused in a circular fashion to record ongoing database changes

Recovery Structures: Redo Log (3)

- Optionally, Oracle may archive files of the online redo log once they fill.
- The online redo log files that are archived are uniquely identified and make up the archived redo log.
- By archiving filled online redo log files, older redo log information is preserved for more extensive database recovery operations, while the pre-allocated online redo log files continue to be reused to store the most current database changes;

Recovery Structures: Rollback Segments

- Rollback segments are used for a number of functions in the operation of an Oracle database.
- In general, the rollback segments of a database store the old values of data changed by uncommitted transactions.
- Among other things, the information in a rollback segment is used during database recovery to "undo" any "uncommitted" changes applied from the redo log to the datafiles.
- Therefore, if database recovery is necessary, the data is in a consistent state after the rollback segments are used to remove all uncommitted data from the datafiles

Recovery Structures: Control Files

- In general, the control file(s) of a database store the status of the physical structure of the database.
- Certain status information in the control file (for example, the current online redo log file, the names of the datafiles, and so on) guides Oracle during instance or media recovery

Online Redo Log

- Every instance of an Oracle database has an associated online redo log to protect the database in case the database experiences an instance failure.
- An online redo log consists of two or more pre-allocated files that store all changes made to the database as they occur.

Online Redo Log (2)

- Online redo log files are filled with *redo entries*.
- Record data used to reconstruct all changes made to the database, including the rollback segments.
- Buffered in a "circular" fashion in the redo log buffer of the SGA and are written to one of the online redo log files by LGWR. Whenever a transaction is committed, LGWR writes the transaction's redo entries from the redo log buffer of the SGA to an online redo log file, and a *system change number* (SCN) is assigned to identify the redo entries for each committed transaction.
- Redo entries can be written to an online redo log file before the corresponding transaction is committed.

Online Redo Log (3)

- Consists of two or more online redo log files (so one is always available for writing while the other is being archived).
- LGWR writes to online redo log files in a circular fashion.
- When the current online redo log file is filled, LGWR begins writing to the next available online redo log file.
- Filled online redo log files are "available" to LGWR for reuse depending on whether archiving is enabled:
    - A filled online redo log file is available once the checkpoint involving the online redo log file has completed.
    - If archiving is enabled, a filled online redo log file is available to LGWR once the checkpoint involving the online redo log file has completed **and** once the file has been archived.

Log Switches

- The point at which Oracle ends writing to one online redo log file and begins writing to another is called a *log switch.*
- A log switch always occurs when the current online redo log file is completely filled and writing must continue to the next online redo log file.
- Oracle assigns each online redo log file a new *log sequence number* every time that a log switch occurs and LGWR begins writing to it.
- If online redo log files are archived, the archived redo log file retains its log sequence number.
- Each redo log file (including online and archived) is uniquely identified by its log sequence number.

Checkpoints

- A *checkpoint* occurs when DBWR writes all the modified database buffers in the SGA, including committed and uncommitted data, to the data files. Checkpoints are implemented for the following reasons:
    - To ensure that blocks in memory that change frequently are written to datafiles regularly. Because of the LRU algorithm of DBWR, a data segment block that changes frequently might never qualify as the least recently used block and thus might never be written to disk if checkpoints did not occur.
    - Because all database changes up to the checkpoint have been recorded in the datafiles, redo log entries before the checkpoint no longer need to be applied to the datafiles if instance recovery is required. Therefore, checkpoints are useful because they can expedite instance recovery.

Checkpoints: When do they occur

- At every log switch.
- LOG_CHECKPOINT_INTERVAL
  LOG_CHECKPOINT_TIMEOUT
- When the beginning of an online tablespace backup is indicated, a checkpoint is forced **only** on the datafiles that constitute the tablespace being backed up.
- If the administrator takes a tablespace offline.
- If the administrator shuts down an instance.
- The administrator can force a database checkpoint to happen on demand.

The mechanics of a Checkpoint

- When a checkpoint occurs, CKPT remembers the location of the next entry to be written in an online redo log file and signals DBWR to write the modified database buffers in the SGA to the datafiles on disk.
- CKPT then updates the headers of all control files and datafiles to reflect the latest checkpoint.
- As a checkpoint proceeds, DBWR writes data to the data files on behalf of both the checkpoint and ongoing database operations. DBWR writes a number of modified data buffers on behalf of the checkpoint, then writes the LRU buffers and then writes more dirty buffers for the checkpoint, and so on, until the checkpoint completes.

The mechanics of a Checkpoint (2)

- A checkpoint can be either "normal" or "fast".
- With a normal checkpoint, DBWR writes a small number of data buffers each time it performs a write on behalf of a checkpoint.
- With a fast checkpoint, DBWR writes a large number of data buffers each time it performs a write on behalf of a checkpoint.

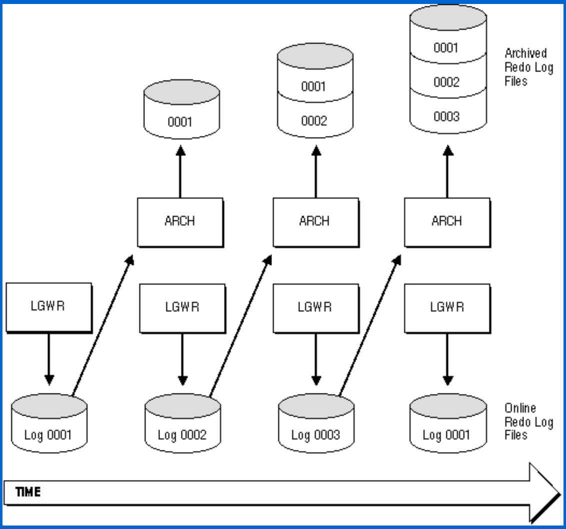
The mechanics of a Checkpoint (3)

- Until a checkpoint completes, all online redo log files written since the last checkpoint are needed in case a database failure interrupts the checkpoint and instance recovery is necessary.
- If LGWR cannot access an online redo log file for writing because a checkpoint has not completed, database operation suspends temporarily until the checkpoint completes and an online redo log file becomes available. In this case, the normal checkpoint becomes a fast checkpoint, so it completes as soon as possible.

Archived Redo Log

- Oracle allows the optional archiving of online redo log files, which creates archived (offline) redo logs. The archiving two key advantages :
  - A database backup, together with online and archived redo log files, guarantees that all committed transactions can be recovered in the event of an operating system or disk failure.
  - Online backups, taken while the database is open and in normal system use, can be used if an archived log is kept permanently.

Automatic Archiving



Control Files

- A control file contains information about a database:
  - the database name
  - the timestamp of database creation
  - the names and locations of associated databases and online redo log files
  - the current log sequence number
  - checkpoint information

Multiplexed Control Files

- Oracle allows multiple, identical control files to be open concurrently and written for the same database.
- If a single disk that contained a control file crashes, the current instance fails when Oracle attempts to access the damaged control file. However, other copies of the current control file are available on different disks, so an instance can be restarted easily without the need for database recovery.
- If *all* control files of a database are permanently lost during operation (several disks fail), the instance is aborted and media recovery is required. Even so, media recovery is not straightforward if an older backup of a control file must be used because a current copy is not available.

Database Backup

- There are two types of backups:
  - Full Backups
  - Partial Backups

Full Backups

- Online: Following a clean shutdown, all of the files that constitute a database are closed and consistent with respect to the current point in time. Thus, a full backup taken after a shutdown can be used to recover to the point in time of the last full backup.
- Offline: A full backup taken while the database is open is not consistent to a given point in time and must be recovered (with the online and archived redo log files) before the database can become available.

Backups and Archiving Mode

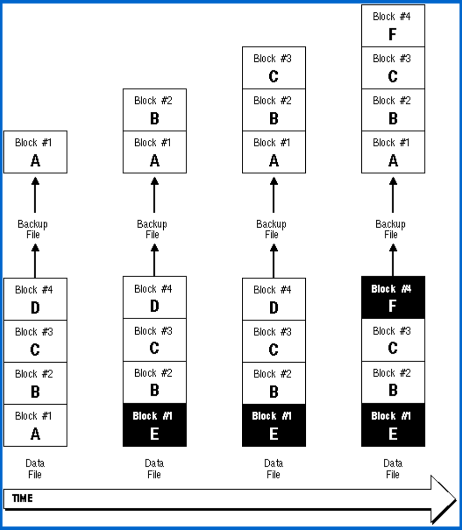
- The datafiles obtained from a full backup are useful in any type of media recovery scheme:
  - If a database is operating in NOARCHIVELOG mode and a disk failure damages some or all of the files that constitute the database, the most recent full backup can be used to *restore* (not recover) the database (all database work performed since the full database backup must be repeated).
  - If a database is operating in ARCHIVELOG mode and a disk failure damages some or all of the files that constitute the database, the datafiles collected by the most recent full backup can be used as part of database *recovery*.

Partial Backups

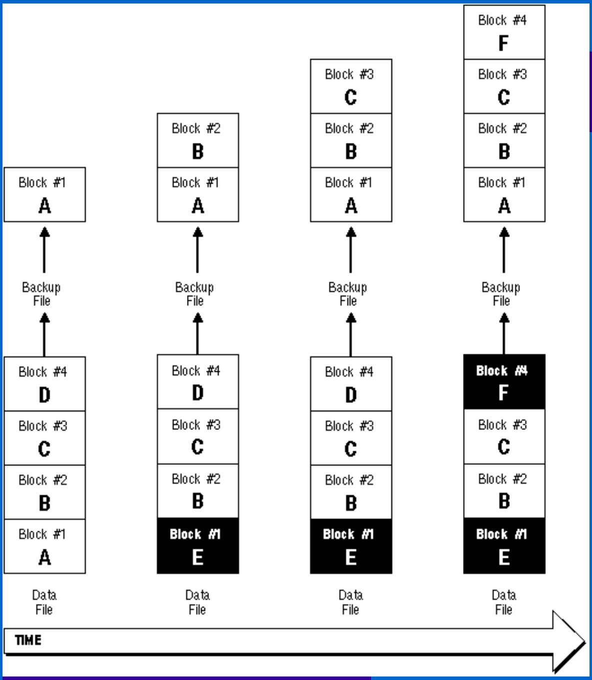
- A *partial backup* is any operating system backup short of a full backup:
  - a backup of all datafiles for an individual tablespace
  - a backup of a single datafile
  - a backup of a control file
- Partial backups are only useful for a database operating in ARCHIVELOG mode. Because an archived redo log is present, the datafiles restored from a partial backup can be made consistent with the rest of the database during recovery procedures.

75

Example of a partial backup of a datafile



76



Recovery Procedures

- Recovering from any type of system failure requires the following:
    - Determining which data structures are intact and which ones need recovery.
    - Following the appropriate recovery steps.
    - Restarting the database so that it can resume normal operations.
    - Ensuring that no work has been lost nor incorrect data entered in the database.

Buffers and DBWR

- Database buffers in the SGA are written to disk only when necessary, using the LRU algorithm. Therefore, datafiles might contain some data blocks modified by uncommitted transactions and some data blocks missing changes from committed transactions.Two potential problems can result if an instance failure occurs:
    - Data blocks modified by a transaction might not be written to the datafiles at commit time and might only appear in the redo log. Therefore, the redo log contains changes that must be reapplied to the database during recovery.
    - Since the redo log might have also contained data that was not committed, the uncommitted transaction changes applied by the redo log (as well as any uncommitted changes applied by earlier redo logs) must be erased from the database.

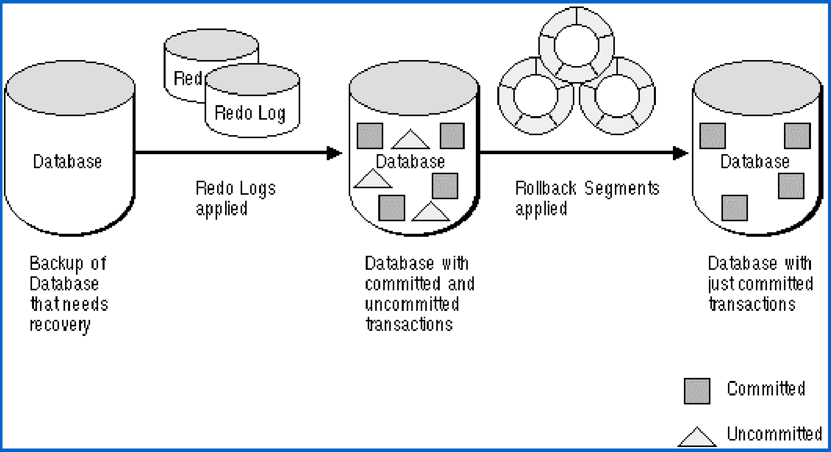
Rolling Forward

- The redo log is a set of operating system files that record all changes made to any database buffer, including data, index, and rollback segments, *whether the changes are committed or uncommitted.*
- The first step of recovery from an instance or disk failure is to *roll forward*, or reapply all of the changes recorded in the redo log. Because rollback data is also recorded in the redo log, rolling forward also regenerates the corresponding rollback segments.
- After roll forward, the data blocks contain all committed changes as well as any uncommitted changes that were recorded in the redo log.

Rolling Back

- Rollback segments record database actions that should be undone during certain database operations.
- In database recovery, rollback segments undo the effects of uncommitted transactions previously applied by the rolling forward phase.

Rolling Forward and rolling back


Redo Log
Redo Log
Database
Redo Logs applied
Database
Rollback Segments applied
Database
Backup of Database that needs recovery
Database with committed and uncommitted transactions
Database with just committed transactions
Committed
Uncommitted

Recovery from Instance Failure

When an instance is aborted, either unexpectedly (for example, an unexpected power outage or a background process failure) or expectedly (for example, when you issue a SHUTDOWN ABORT or STARTUP FORCE statement), instance failure occurs, and instance recovery is required. Instance recovery restores a database to its transaction-consistent state just before instance failure.

Steps in Recovery from Instance Failure

- Roll forward to recover data that has not been recorded in the datafiles, yet has been recorded in the online redo log, including the contents of rollback segments.
- Open the database. Instead of waiting for all transactions to be rolled back before making the database available, Oracle enables the database to be opened as soon as cache recovery is complete. Any data that is not locked by unrecovered transactions is immediately available. This feature is called *fast warmstart*.
- Mark all transactions system-wide that were active at the time of failure as DEAD and mark the rollback segments containing these transactions as PARTIALLY AVAILABLE.
- Recover dead transactions as part of SMON recovery.
- Resolve any pending distributed transactions undergoing a two-phase commit at the time of the instance failure.

Recovery from Media Failure

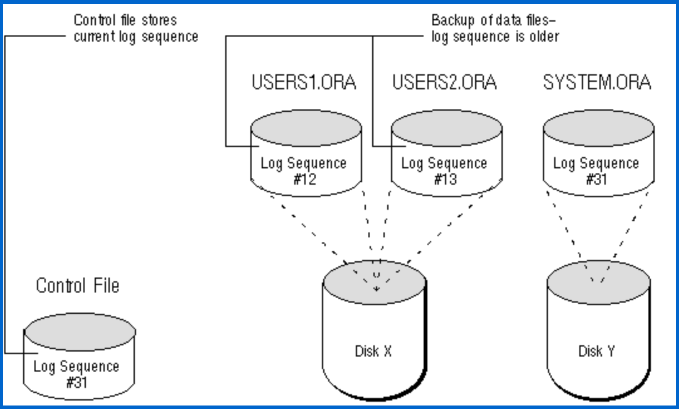
- Recovery from a media failure can take two forms:
  - If the online redo log is not archived, recovery from a media failure is a simple restoration of the most current full backup. All work performed after the full backup was taken must be redone manually.
  - Otherwise, recovery from a media failure can be an actual recovery procedure, to reconstruct the damaged database to a specified transaction-consistent state before the media failure.
- Recovery from a media failure, no matter what form, always recovers the entire database to a transaction-consistent state before the media failure.

Example of Complete Media Recovery

- Assume the following:
  - the database has three datafiles: USERS1 and USERS2 are datafiles that constitute the USERS tablespace, stored on Disk X of the database server, SYSTEM is the datafile that constitutes the SYSTEM tablespace, stored on Disk Y of the database server.
  - Disk X of the database server has crashed.
  - the online redo log file being written to at the time of the disk failure has a log sequence number of 31.
  - the database is in ARCHIVELOG mode
- Recovery of the two datafiles that constitute the USERS tablespace is necessary.

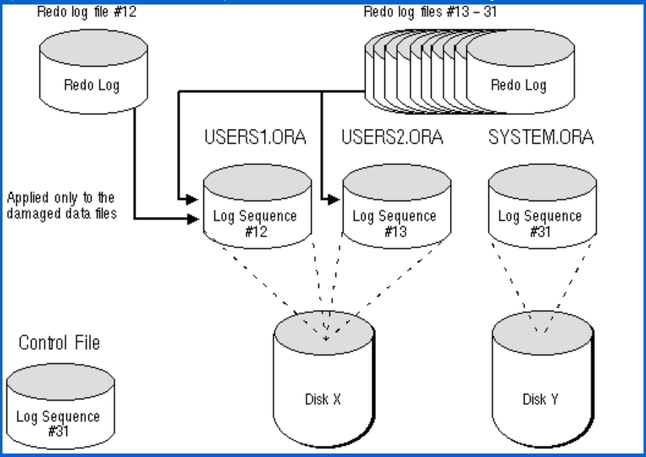
Complete Media Recovery (2)

Phase 1: Restoration of Backup Datafiles After Disk X has been repaired, the most recent backup files are used to restore only the damaged datafiles USERS1 and USERS2.



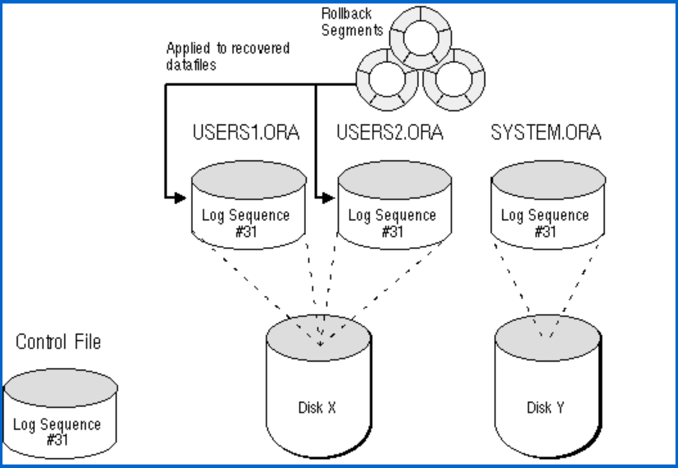
Complete Media Recovery (3)

Phase 2: Rolling Forward with the Redo Log Oracle applies redo log files (archived and online) to datafiles, as necessary.



Complete Media Recovery (4)

Phase 3: Rolling Back Using Rollback Segments



Incomplete Media Recovery

In specific situations (for example, the loss of all active online redo log files, or a user error, such as the accidental dropping of an important table), complete media recovery may not be possible or may not be desired. In such situations, *incomplete media recovery* is performed to reconstruct the damaged database to a transaction consistent state before the media failure or user error.

Incomplete Media Recovery

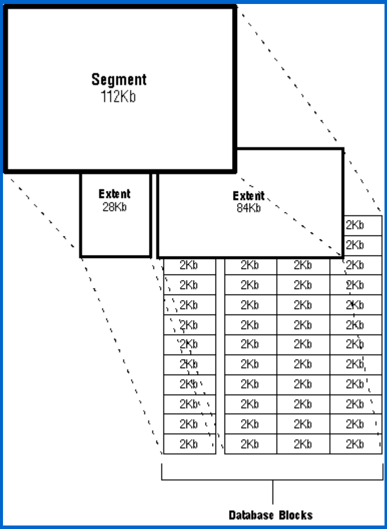
There are different types of incomplete media recovery that might be used, depending on the situation that requires incomplete media recovery

- *cancel-based* recovery is used when one or more redo logs (online or archived) have been damaged by a media failure and are not available for required recovery procedures.
- *Time-Based and Change-Based Recovery* Incomplete media recovery is desirable if the database administrator would like to recover to a specific point in the past.

Part III

Database Structures

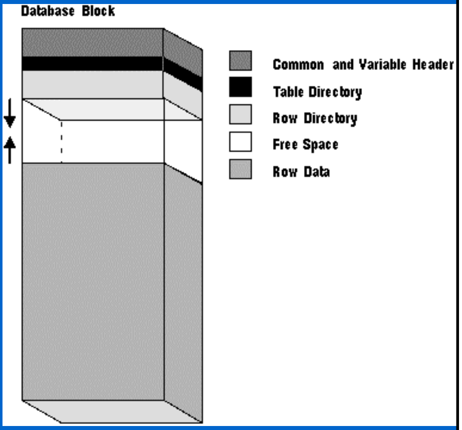
Data Blocks, Extents and Segments



Data Blocks

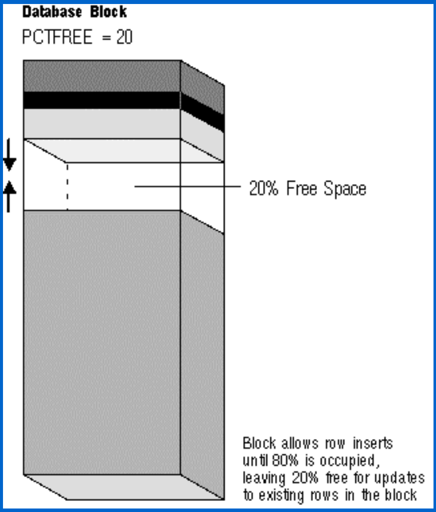
- **Header**: general block info: block address and the type of segment; for example, data, index, or rollback.
- **Table Directory:** info about the tables having

  rows in this block.
- **Row Directory:** row info about the actual rows in the block
- **Row Data**



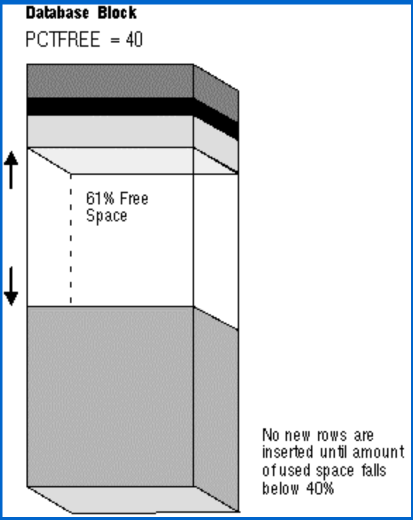
PCTFREE

The PCTFREE parameter is used to set the percentage of a block to be reserved (kept free) for possible updates to rows that already are contained in that block.

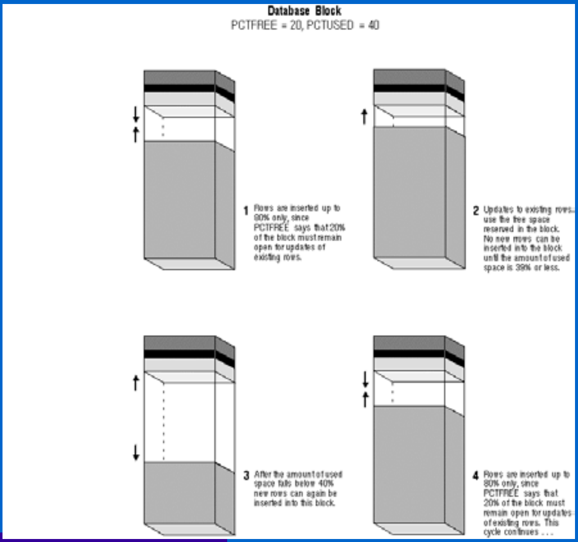

Database Block
PCTFREE = 20
20% Free Space
Block allows row inserts until 80% is occupied, leaving 20% free for updates to existing rows in the block

PCTUSED

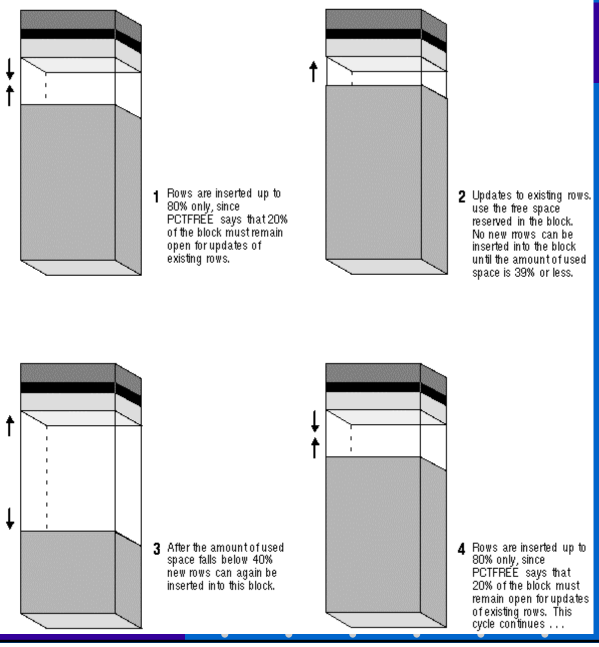
- After a data block becomes full, as determined by PCTFREE, Oracle does not consider the block for the insertion of new rows until the percentage of the block being used falls below the parameter PCTUSED. Before this value is achieved, Oracle uses the free space of the data block only for updates to rows already contained in the data block.


Database Block
PCTFREE = 40
61% Free Space
No new rows are inserted until amount of used space falls below 40%

PCTFREE and PCTUSED


Database Block
PCTFREE = 20, PCTUSED = 40
1 Rows are inserted up to 80% only, since PCTFREE says that 20% of the block must remain open for updates of existing rows.
2 Updates to existing rows use the free space reserved in the block. No new rows can be inserted into the block until the amount of used space is 39% or less.
3 After the amount of used space falls below 40% new rows can again be inserted into this block.
4 Rows are inserted up to 80% only, since PCTFREE says that 20% of the block must remain open for updates of existing rows. This cycle continues . . .

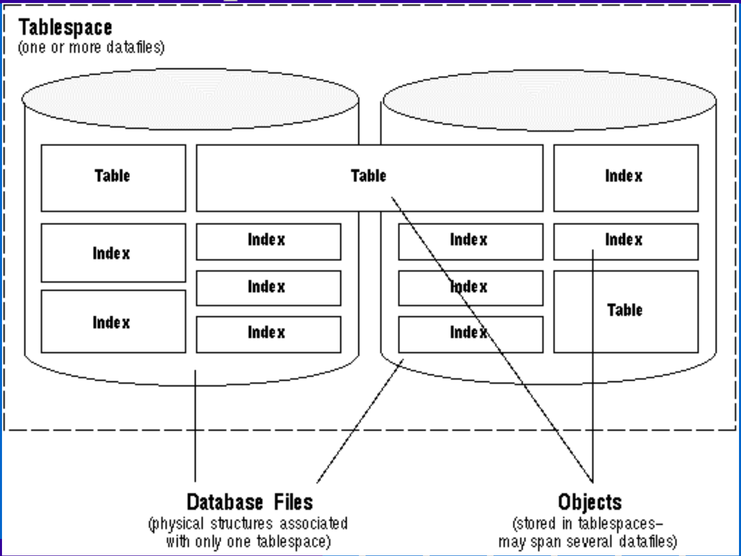
Database Block
PCTFREE = 20, PCTUSED = 40


1 Rows are inserted up to 80% only, since PCTFREE says that 20% of the block must remain open for updates of existing rows.
2 Updates to existing rows. use the free space reserved in the block. No new rows can be inserted into the block until the amount of used space is 39% or less.
3 After the amount of used space falls below 40% new rows can again be inserted into this block.
4 Rows are inserted up to 80% only, since PCTFREE says that 20% of the block must remain open for updates of existing rows. This cycle continues . . .

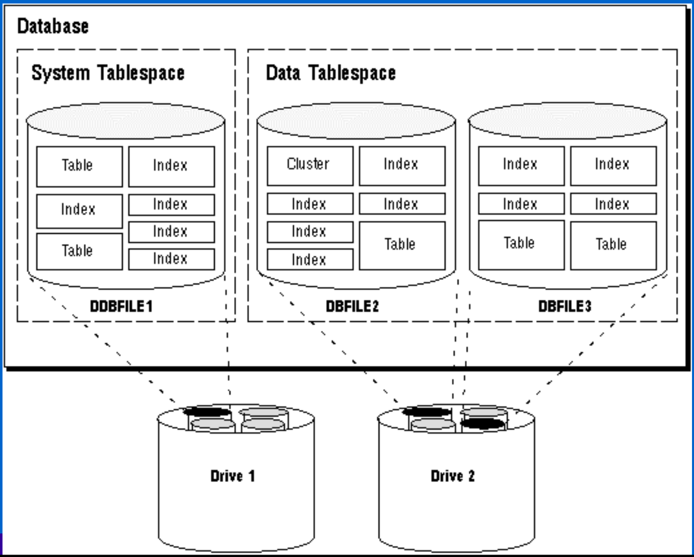
Segments

- There are four types of segments used in Oracle databases:
  - data segments
  - index segments
  - rollback segments
  - temporary segments

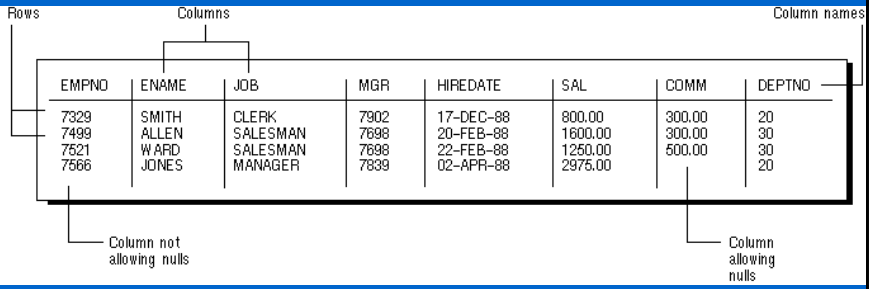
Tablespace and Datafiles



Databases



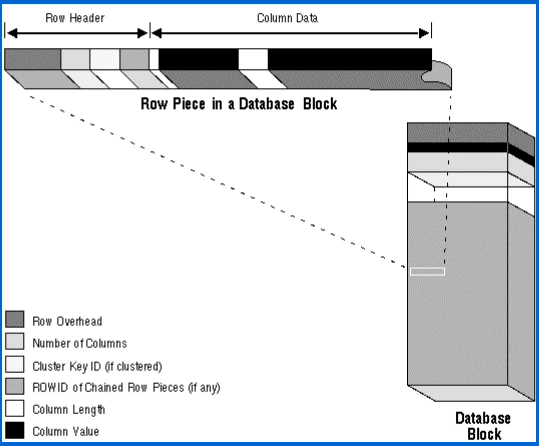
Tables

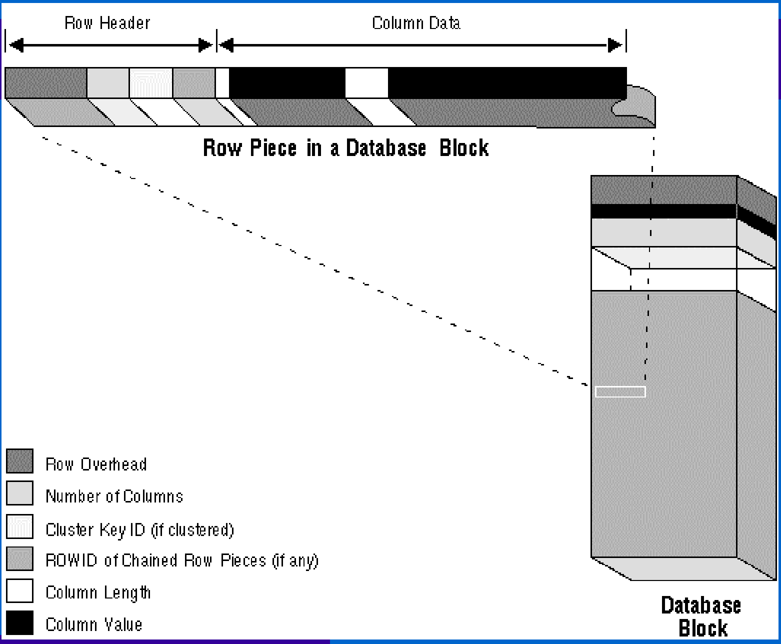

Rows
Columns
Column names
Column not allowing nulls
Column allowing nulls

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7329	SMITH	CLERK	7902	17-DEC-88	800.00	300.00	20
7499	ALLEN	SALESMAN	7698	20-FEB-88	1600.00	300.00	30
7521	WARD	SALESMAN	7698	22-FEB-88	1250.00	500.00	30
7566	JONES	MANAGER	7839	02-APR-88	2975.00		20

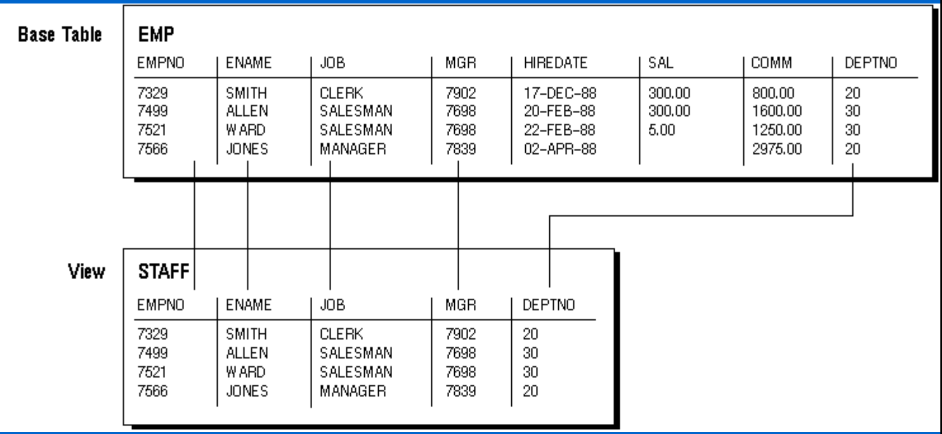
Row Format and Sizes

Oracle stores each row of a database table as one or more row pieces


Row Header
Column Data
Row Piece in a Database Block
Database Block
▨ Row Overhead
□ Number of Columns
□ Cluster Key ID (if clustered)
▨ ROWID of Chained Row Pieces (if any)
□ Column Length
■ Column Value


Row Header
Column Data
Row Piece in a Database Block
Database Block
Row Overhead
Number of Columns
Cluster Key ID (if clustered)
ROWID of Chained Row Pieces (if any)
Column Length
Column Value

Views



EMP

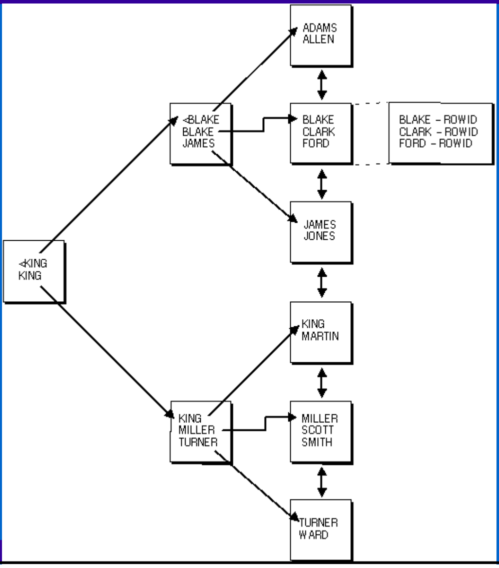
EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7329	SMITH	CLERK	7902	17-DEC-88	300.00	800.00	20
7499	ALLEN	SALESMAN	7698	20-FEB-88	300.00	1600.00	30
7521	WARD	SALESMAN	7698	22-FEB-88	5.00	1250.00	30
7566	JONES	MANAGER	7839	02-APR-88		2975.00	20

STAFF

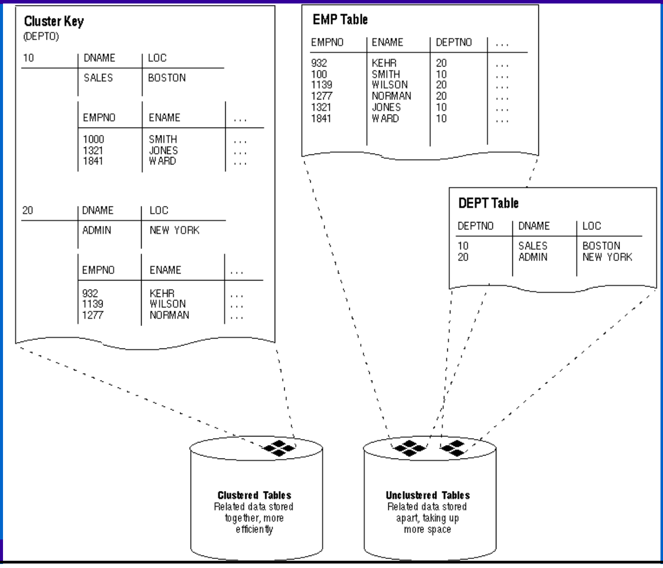
EMPNO	ENAME	JOB	MGR	DEPTNO
7329	SMITH	CLERK	7902	20
7499	ALLEN	SALESMAN	7698	30
7521	WARD	SALESMAN	7698	30
7566	JONES	MANAGER	7839	20

Indexes

Oracle uses B*-tree indexes that are balanced to equalize access times to any row



Clusters


Cluster Key (DEPTO)

10	DNAME	LOC
	SALES	BOSTON

EMPNO	ENAME	. . .

1000
1321
1841
	SMITH
	JONES
	WARD
		. . .
		. . .
		. . .

20	DNAME	LOC
	ADMIN	NEW YORK

EMPNO	ENAME	. . .

932
1139
1277
	KEHR
	WILSON
	NORMAN
		. . .
		. . .
		. . .

EMP Table

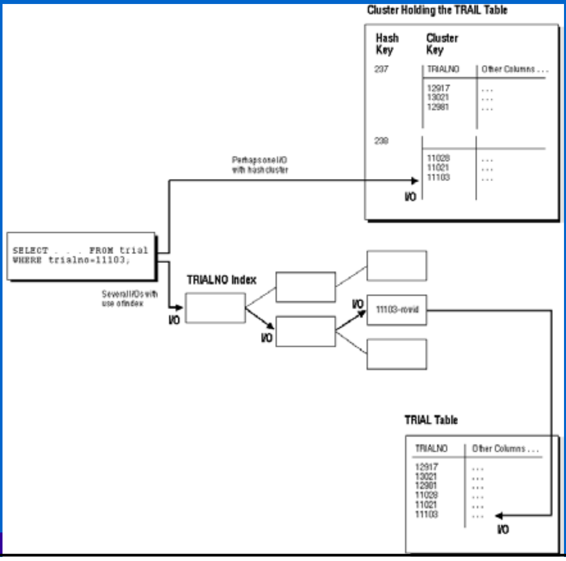
EMPNO	ENAME	DEPTNO	. . .
932	KEHR	20	. . .
100	SMITH	10	. . .
1139	WILSON	20	. . .
1277	NORMAN	20	. . .
1321	JONES	10	. . .
1841	WARD	10	. . .

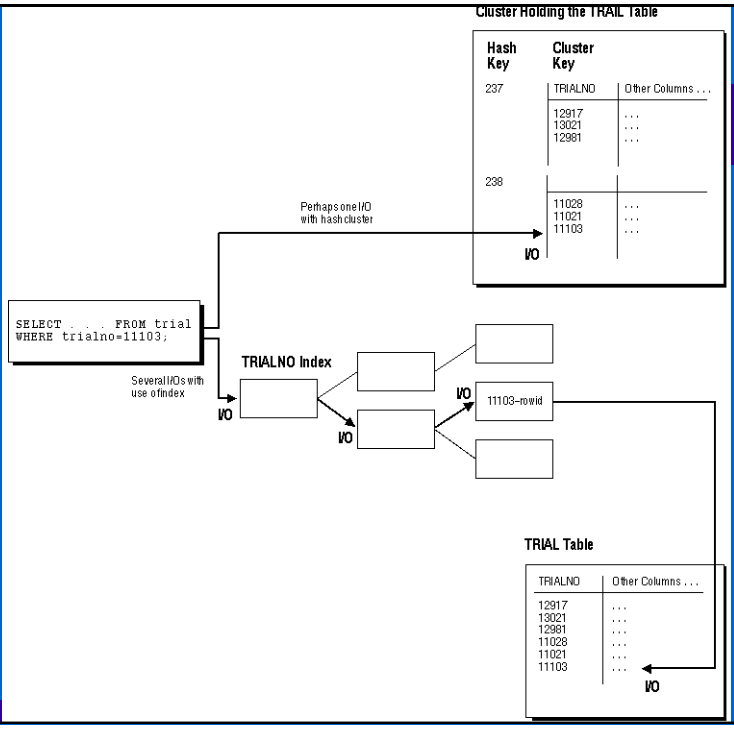
DEPT Table

DEPTNO	DNAME	LOC
10	SALES	BOSTON
20	ADMIN	NEW YORK

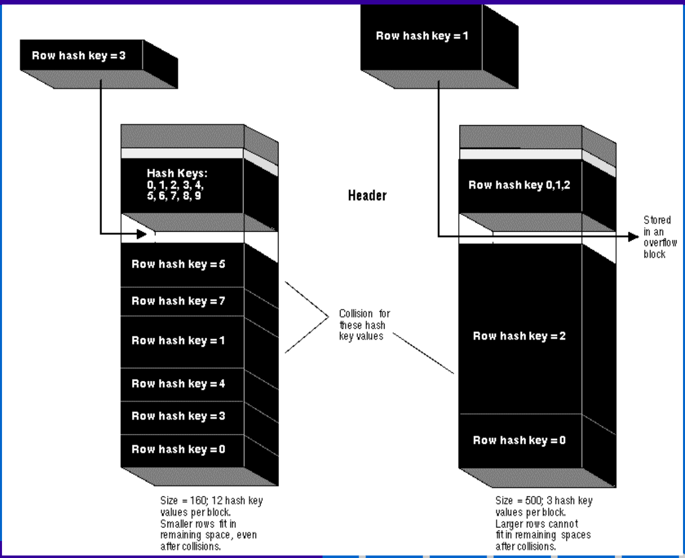
Clustered Tables
Related data stored together, more efficiently
Unclustered Tables
Related data stored apart, taking up more space

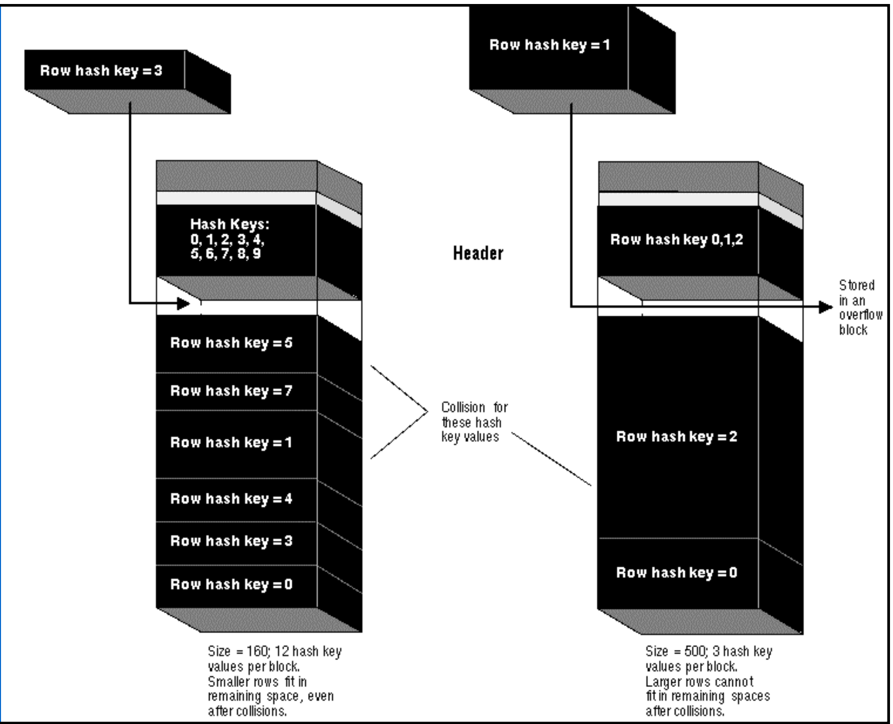
Hash Clusters





Allocation of space in Hash Clusters


Row hash key = 3
Row hash key = 1
Hash Keys: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Header
Row hash key 0,1,2
Stored in an overflow block
Row hash key = 5
Row hash key = 7
Row hash key = 1
Row hash key = 4
Row hash key = 3
Row hash key = 0
Collision for these hash key values
Row hash key = 2
Row hash key = 0
Size = 160; 12 hash key values per block. Smaller rows fit in remaining space, even after collisions.
Size = 500; 3 hash key values per block. Larger rows cannot fit in remaining spaces after collisions.


Row hash key = 3
Row hash key = 1
Hash Keys: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Header
Row hash key 0,1,2
Stored in an overflow block
Row hash key = 5
Row hash key = 7
Row hash key = 1
Row hash key = 4
Row hash key = 3
Row hash key = 0
Collision for these hash key values
Row hash key = 2
Row hash key = 0
Size = 160; 12 hash key values per block. Smaller rows fit in remaining space, even after collisions.
Size = 500; 3 hash key values per block. Larger rows cannot fit in remaining spaces after collisions.

Datatypes

Data Type	Description	Column Length (bytes)
CHAR (size)	Fixed-length character data of length size.	Fixed for every row in the table (with trailing blanks);
VARCHAR2 (size)	Variable-length character data.	Variable for each row, up to 2000 bytes per row.
NUMBER (p, s)	Variable-length numeric data.	Variable for each row. The maximum space required for a given column is 21 bytes per row.
DATE	Fixed-length date and time data	Fixed at seven bytes for each row in the table.
LONG	Variable-length character data.	Variable for each row in the table, up to 2^31 - 1 bytes, or two gigabytes, per row.
RAW (size)	Variable-length raw binary data	Variable for each row in the table, up to 255 bytes per row.
LONG RAW	Variable-length raw binary data.	Variable for each row in the table, up to 2^31 - 1 bytes, or two gigabytes, per row.
ROWID	Binary data representing row addresses.	Fixed at six bytes for each row in the table.
MLSLABEL	Variable-length binary data representing operating system labels.	Variable for each row in the table, ranging from two to five bytes per row.

Data Integrity

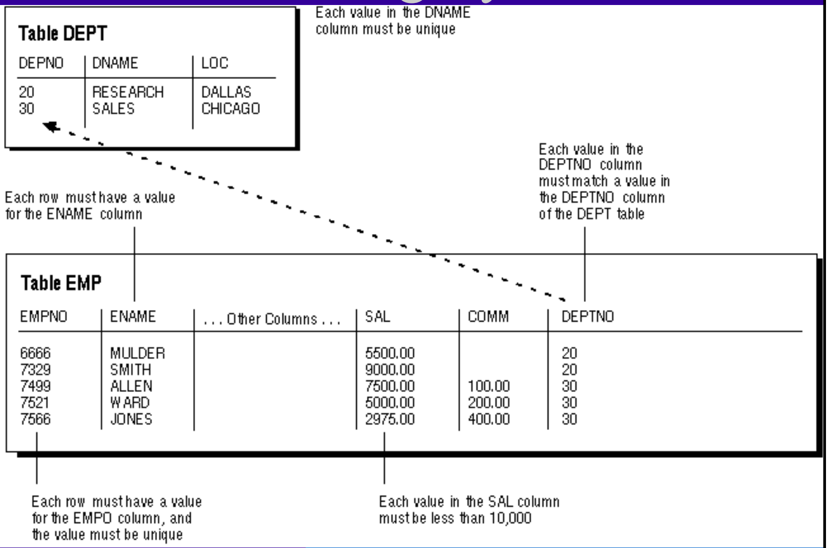


Table DEPT

DEPTNO	DNAME	LOC
20	RESEARCH	DALLAS
30	SALES	CHICAGO

Table EMP

EMPNO	ENAME	. . . Other Columns . . .	SAL	COMM	DEPTNO
6666	MULDER		5500.00		20
7329	SMITH		9000.00		20
7499	ALLEN		7500.00	100.00	30
7521	WARD		5000.00	200.00	30
7566	JONES		2975.00	400.00	30

Each value in the DNAME column must be unique
Each row must have a value for the ENAME column
Each value in the DEPTNO column must match a value in the DEPTNO column of the DEPT table
Each row must have a value for the EMPO column, and the value must be unique
Each value in the SAL column must be less than 10,000

Not Null Integrity Constraint

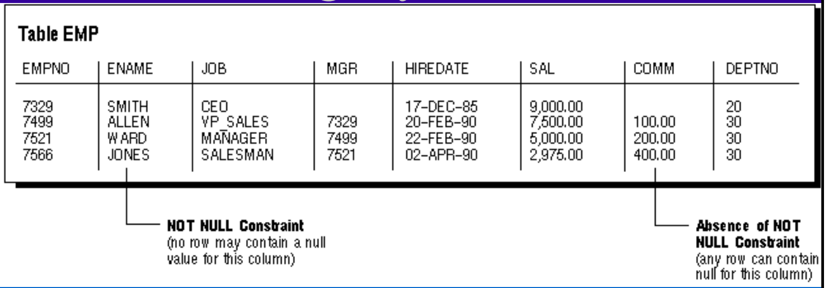


Table EMP

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7329	SMITH	CEO		17-DEC-85	9,000.00		20
7499	ALLEN	VP_SALES	7329	20-FEB-90	7,500.00	100.00	30
7521	WARD	MANAGER	7499	22-FEB-90	5,000.00	200.00	30
7566	JONES	SALESMAN	7521	02-APR-90	2,975.00	400.00	30

NOT NULL Constraint
(no row may contain a null value for this column)
Absence of NOT NULL Constraint
(any row can contain null for this column)

Unique Key

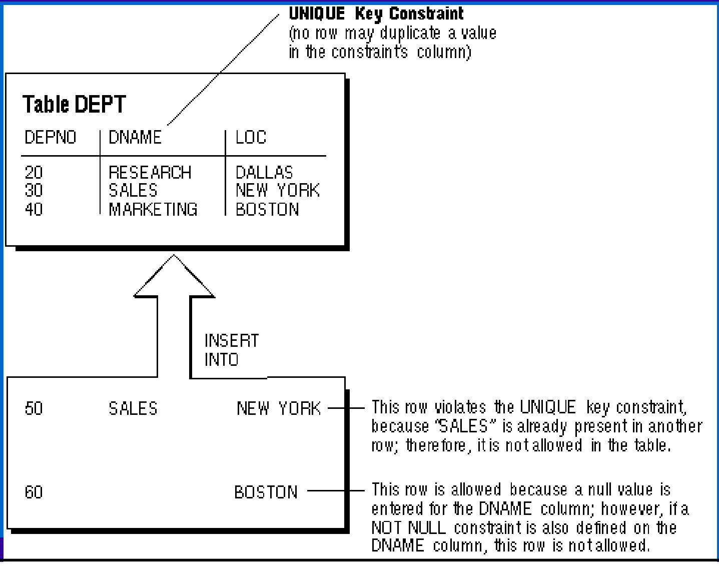


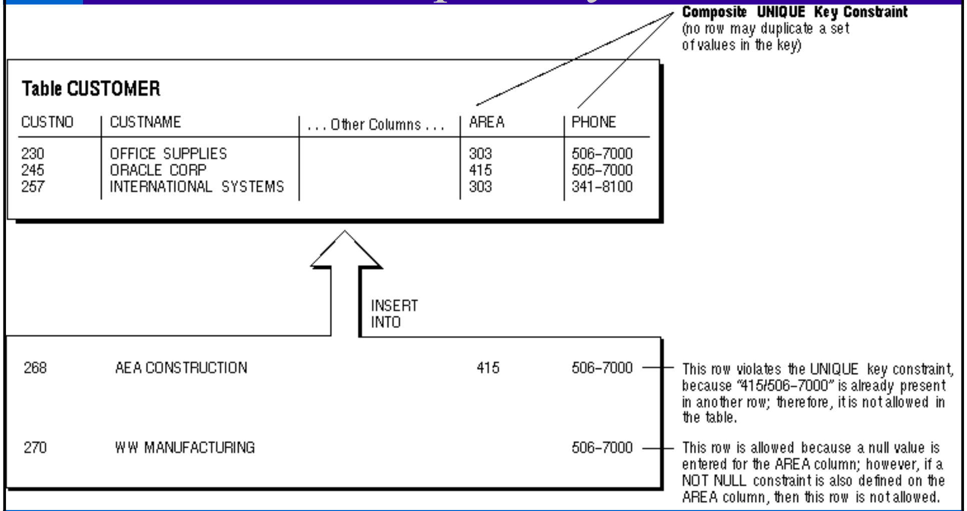
Table DEPT

DEPNO	DNAME	LOC
20	RESEARCH	DALLAS
30	SALES	NEW YORK
40	MARKETING	BOSTON

50	SALES	NEW YORK
60		BOSTON

UNIQUE Key Constraint
(no row may duplicate a value in the constraint's column)
INSERT INTO
This row violates the UNIQUE key constraint, because "SALES" is already present in another row; therefore, it is not allowed in the table.
This row is allowed because a null value is entered for the DNAME column; however, if a NOT NULL constraint is also defined on the DNAME column, this row is not allowed.

Unique Keys



Composite UNIQUE Key Constraint
(no row may duplicate a set of values in the key)

Table CUSTOMER

CUSTNO	CUSTNAME	. . . Other Columns . . .	AREA	PHONE
230	OFFICE SUPPLIES		303	506-7000
245	ORACLE CORP		415	505-7000
257	INTERNATIONAL SYSTEMS		303	341-8100

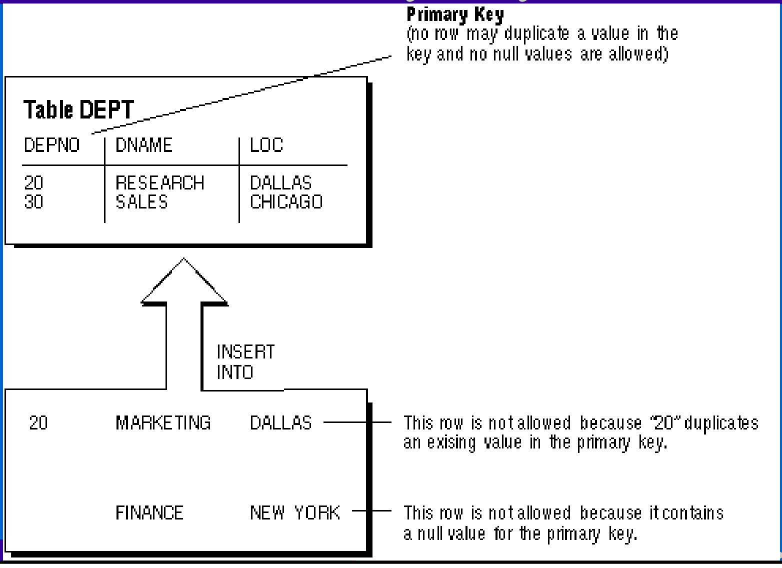
INSERT INTO

268	AEA CONSTRUCTION	415	506-7000
270	WW MANUFACTURING		506-7000

This row violates the UNIQUE key constraint, because "415/506-7000" is already present in another row; therefore, it is not allowed in the table.

This row is allowed because a null value is entered for the AREA column; however, if a NOT NULL constraint is also defined on the AREA column, then this row is not allowed.

Primary Key



Primary Key
(no row may duplicate a value in the key and no null values are allowed)

Table DEPT

DEPNO	DNAME	LOC
20	RESEARCH	DALLAS
30	SALES	CHICAGO

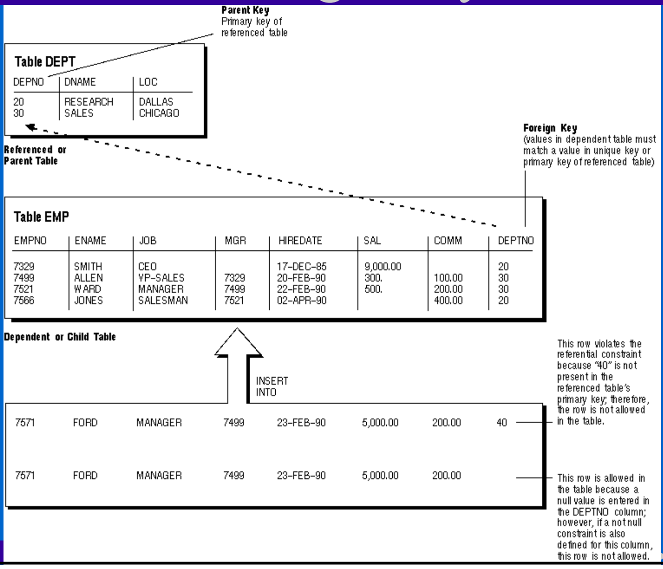
INSERT INTO

20	MARKETING	DALLAS
	FINANCE	NEW YORK

This row is not allowed because "20" duplicates an exising value in the primary key.

This row is not allowed because it contains a null value for the primary key.

Foreign Key



Parent Key
Primary key of referenced table

Table DEPT

DEPNO	DNAME	LOC
20	RESEARCH	DALLAS
30	SALES	CHICAGO

Referenced or Parent Table

Foreign Key
(values in dependent table must match a value in unique key or primary key of referenced table)

Table EMP

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7329	SMITH	CEO		17-DEC-85	9,000.00		20
7499	ALLEN	VP-SALES	7329	20-FEB-90	300.	100.00	30
7521	WARD	MANAGER	7499	22-FEB-90	500.	200.00	30
7566	JONES	SALESMAN	7521	02-APR-90		400.00	20

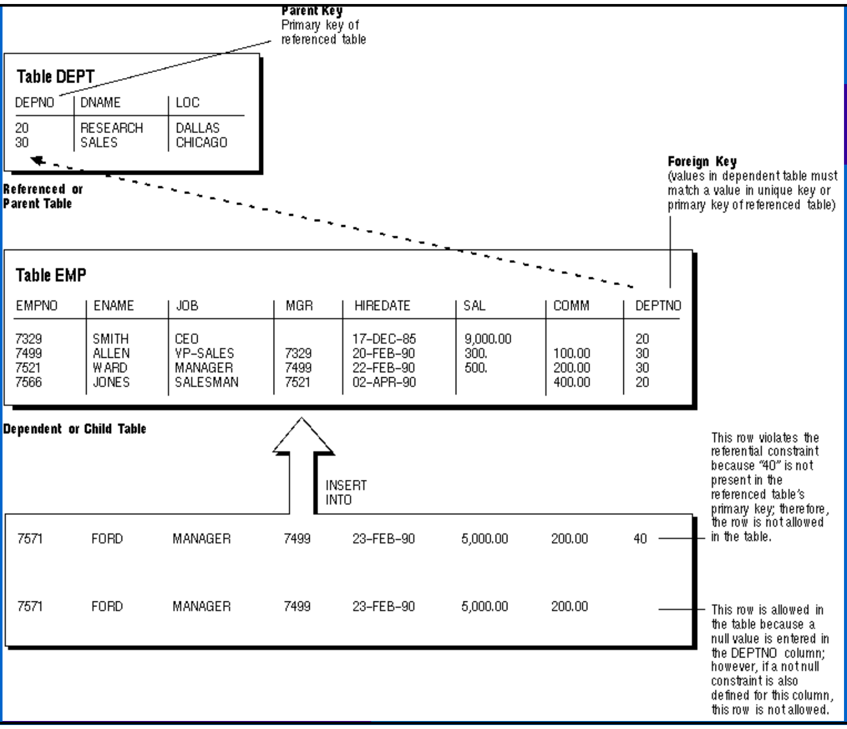
Dependent or Child Table

INSERT INTO

7571	FORD	MANAGER	7499	23-FEB-90	5,000.00	200.00	40
7571	FORD	MANAGER	7499	23-FEB-90	5,000.00	200.00	

This row violates the referential constraint because "40" is not present in the referenced table's primary key; therefore, the row is not allowed in the table.

This row is allowed in the table because a null value is entered in the DEPTNO column; however, if a not null constraint is also defined for this column, this row is not allowed.



Parent Key
Primary key of referenced table

Table DEPT

DEPNO | DNAME | LOC
20 | RESEARCH | DALLAS
30 | SALES | CHICAGO

Referenced or Parent Table

Foreign Key
(values in dependent table must match a value in unique key or primary key of referenced table)

Table EMP

EMPNO | ENAME | JOB | MGR | HIREDATE | SAL | COMM | DEPTNO
7329 | SMITH | CEO | | 17-DEC-85 | 9,000.00 | | 20
7499 | ALLEN | VP-SALES | 7329 | 20-FEB-90 | 300. | 100.00 | 30
7521 | WARD | MANAGER | 7499 | 22-FEB-90 | 500. | 200.00 | 30
7566 | JONES | SALESMAN | 7521 | 02-APR-90 | | 400.00 | 20

Dependent or Child Table

INSERT INTO

7571 | FORD | MANAGER | 7499 | 23-FEB-90 | 5,000.00 | 200.00 | 40
7571 | FORD | MANAGER | 7499 | 23-FEB-90 | 5,000.00 | 200.00 |

This row violates the referential constraint because "40" is not present in the referenced table's primary key; therefore, the row is not allowed in the table.

This row is allowed in the table because a null value is entered in the DEPTNO column; however, if a not null constraint is also defined for this column, this row is not allowed.

Self-referential constraint

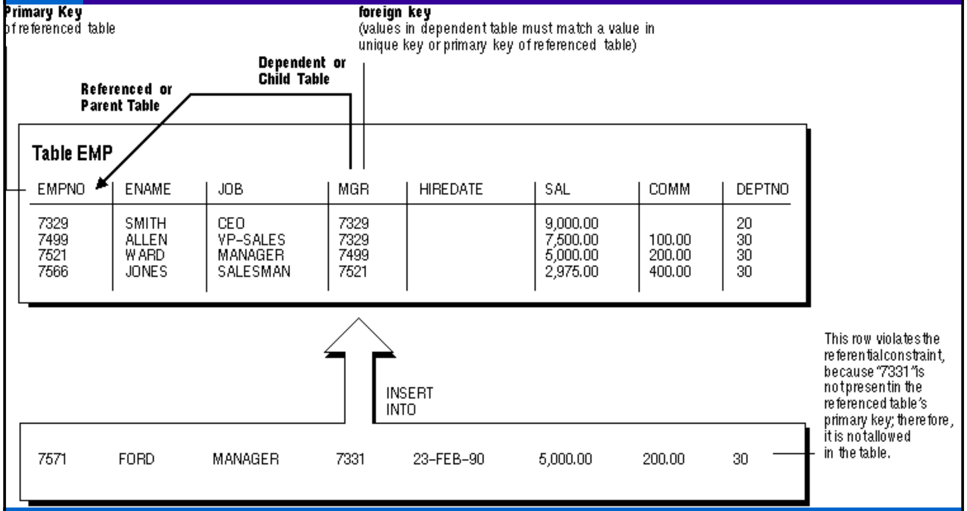

Primary Key
of referenced table
foreign key
(values in dependent table must match a value in unique key or primary key of referenced table)
Dependent or Child Table
Referenced or Parent Table
INSERT INTO

Table EMP

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7329	SMITH	CEO	7329		9,000.00		20
7499	ALLEN	VP-SALES	7329		7,500.00	100.00	30
7521	WARD	MANAGER	7499		5,000.00	200.00	30
7566	JONES	SALESMAN	7521		2,975.00	400.00	30

7571	FORD	MANAGER	7331	23-FEB-90	5,000.00	200.00	30

This row violates the referential constraint, because "7331" is not present in the referenced table's primary key; therefore, it is not allowed in the table.

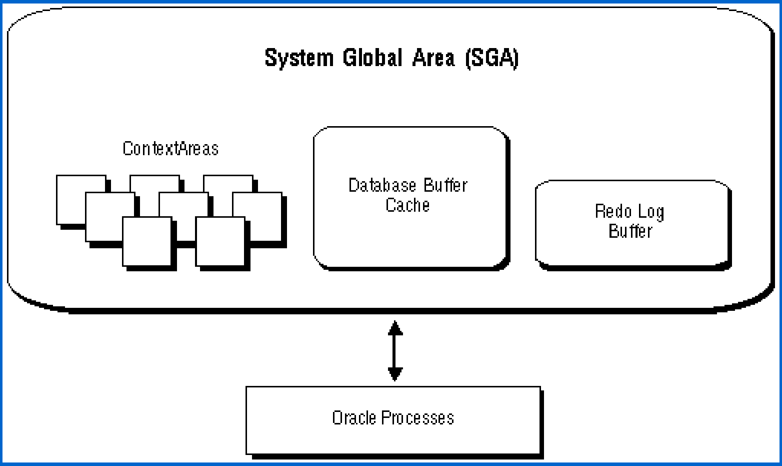
Part IV

System Architecture

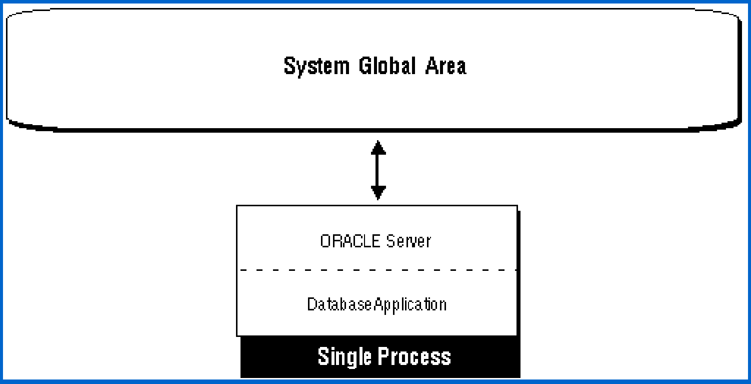
An Oracle Instance

- Regardless of the type of computer executing Oracle and the particular memory and process options being used, every running Oracle database is associated with an Oracle instance.
- Every time a database is started on a database server, Oracle allocates a memory area called the System Global Area (SGA) and starts one or more Oracle processes.
- The combination of the SGA and the Oracle processes is called an Oracle database instance.
- Oracle starts an instance, then mounts a database to the instance.

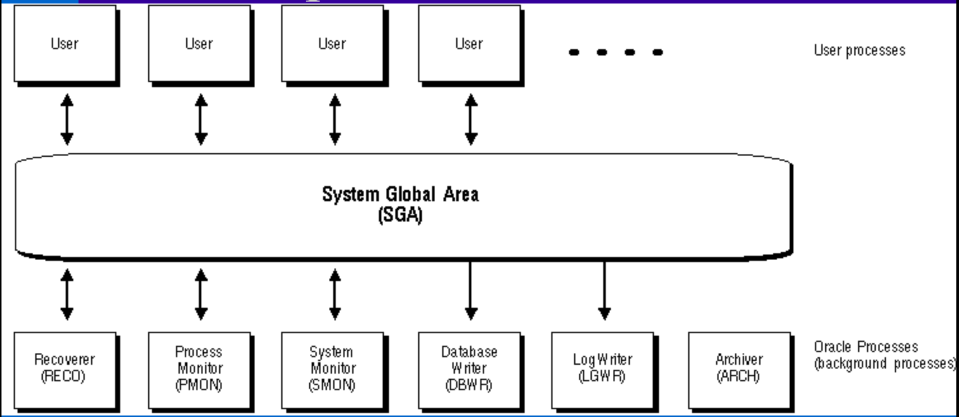
An Oracle Instance (2)



Single Process Instance



Multiple Process Instance



DBWR

- When a buffer in the buffer cache is modified, it is marked "dirty". As buffers are filled and dirtied by user processes, the number of free buffers diminishes. If the number of free buffers drops too low, user processes that must read blocks from disk into the cache are not able to find free buffers. DBWR manages the buffer cache so that user processes can always find free buffers.
- LRU keeps the most recently used data blocks in memory and thus minimizes I/O. DBWR keeps blocks that are used often, for example, blocks that are part of frequently accessed small tables or indexes, in the cache so that they do not need to be read in again from disk. DBWR removes blocks that are accessed infrequently (for example, blocks that are part of very large tables or leaf blocks from very large indexes) from the SGA.
- If the DBWR process becomes too active, it may write blocks to disk that are about to be needed again.

DBWR (2)

The DBWR process writes dirty buffers to disk under the following conditions:

- When a server process moves a buffer to the dirty list and discovers that the dirty list has reached a threshold length, the server process signals DBWR to write.
- When a server process searches a threshold limit of buffers in the LRU list without finding a free buffer, it stops searching and signals DBWR to write.
- When a time-out occurs (every three seconds), DBWR signals itself.
- When a checkpoint occurs, the Log Writer process (LGWR) signals DBWR.

LGWR

The *Log Writer process (LGWR)* writes the redo log buffer to a redo log file on disk. LGWR writes all redo entries that have been copied into the buffer since the last time it wrote. LGWR writes one contiguous portion of the buffer to disk. LGWR writes

- a commit record when a user process commits a transaction
- redo buffers every three seconds
- redo buffers when the redo log buffer is one-third full
- redo buffers when the DBWR process writes modified buffers to disk

CKPT

- When a checkpoint occurs, Oracle must update the headers of all datafiles to indicate the checkpoint.
- In normal situations, this job is performed by LGWR.
- However, if checkpoints significantly degrade system performance (usually, when there are many datafiles), it is possible to enable the Checkpoint process (CKPT) to separate the work of performing a checkpoint from other work performed by LGWR, the Log Writer process (LGWR).

SMON, PMON

- *SMON* performs instance recovery at instance start up. SMON is also responsible for cleaning up temporary segments that are no longer in use; it also coalesces contiguous free extents to make larger blocks of free space available.
- *PMON* performs process recovery when a user process fails. PMON is responsible for cleaning up the cache and freeing resources that the process was using.

Other Processes

- Reco: Distributed recovery
- Arch: The archiver of redo log files
- Lock0: Inter instance locking
- SPN0: Snapshot refresh
- Dnnn: Dispatcher processes

Memory Structures

Oracle uses memory to store the following information:

- program code being executed
- information about a connected session, even if it is not currently active
- data needed during program execution (for example, the current state of a query from which rows are being fetched)
- information that is shared and communicated among Oracle processes (for example, locking information)
- cached information that is also permanently stored on peripheral memory (for example, a data block)

Memory Structures (2)

The basic memory structures associated with Oracle include:

- software code areas
- the system global area (SGA)
  - the database buffer cache
  - the redo log buffer
  - the shared pool
- program global areas (PGA)
  - stack areas
  - data areas
- sort areas

Software Code Areas

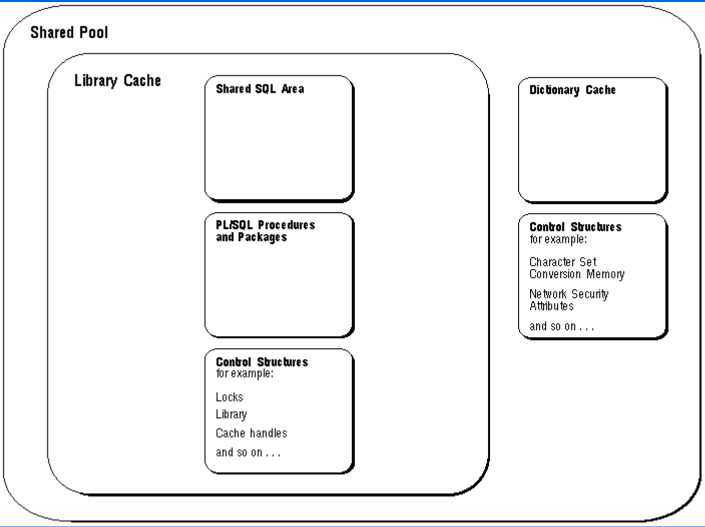
Software code areas are portions of memory used to store code that is being or may be executed. The code for Oracle is stored in a software area, which is typically at a location different from users' programs -- a more exclusive or protected location.

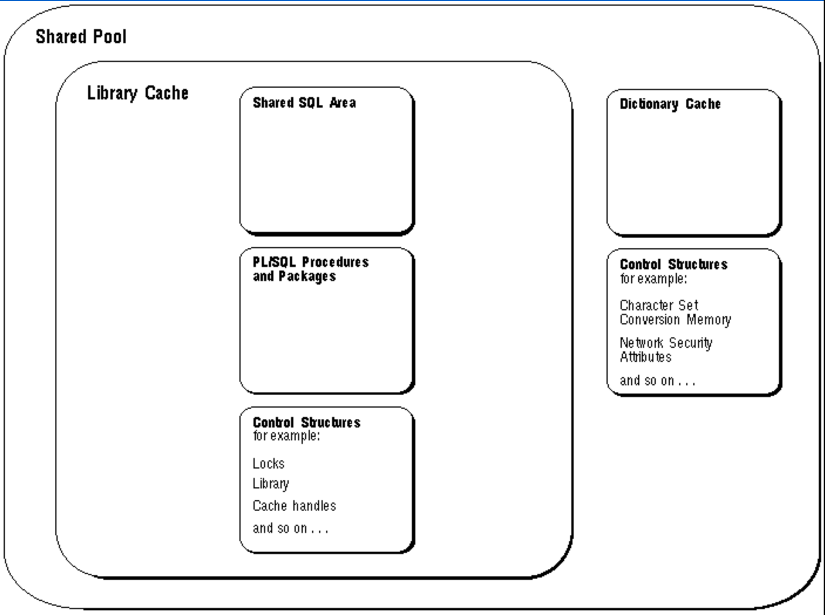
SGA

A System Global Area (SGA) is a group of shared memory structures that contain data and control information for one Oracle database instance. The SGA contains the following subdivisions:

- the database buffer cache
- the redo log buffer
- the shared pool
- the data dictionary cache
- other miscellaneous information

SGA: The Shared Pool

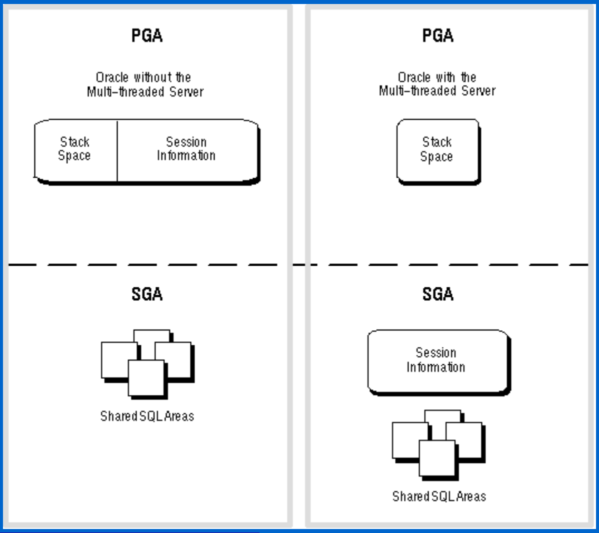

Shared Pool
Library Cache
Shared SQL Area
PL/SQL Procedures and Packages
Control Structures for example:
Locks
Library
Cache handles
and so on . . .
Dictionary Cache
Control Structures for example:
Character Set Conversion Memory
Network Security Attributes
and so on . . .


Shared Pool
Library Cache
Shared SQL Area
PL/SQL Procedures and Packages
Control Structures for example:
Locks
Library
Cache handles
and so on . . .
Dictionary Cache
Control Structures for example:
Character Set Conversion Memory
Network Security Attributes
and so on . . .

The Shared Pool (2)

- Oracle represents each SQL statement it executes with a *shared SQL area* and a *private SQL area*. Oracle recognizes when two users are executing the same SQL statement and reuses the same shared part for those users. However, each user must have a separate copy of the statement's private SQL area.
- A shared SQL area is a memory area that contains the parse tree and execution plan for a single SQL statement.
- A private SQL area is a memory area that contains data such as bind information and runtime buffers.

Program Global Area (PGA)



Sort Areas

- Portions of memory in which Oracle sorts data are called *sort areas.*
- If the amount of data to be sorted does not fit into a sort area, then the data is divided into smaller pieces that do fit. Each piece is then sorted individually. The individual sorted pieces are called "runs". After sorting all the runs, Oracle merges them to produce the final result.

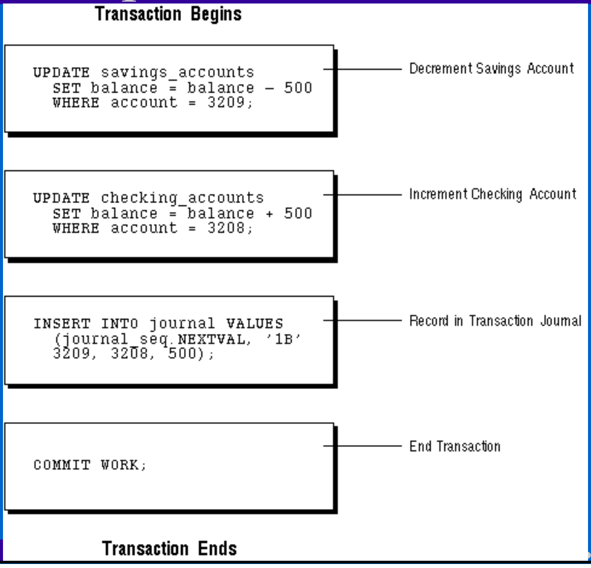
Part V

Concurrency Control

Transactions

- A *transaction* is a logical unit of work that contains one or more SQL statements.
- A transaction is an atomic unit; the effects of all the SQL statements in a transaction can be either all *committed* (applied to the database) or all *rolled back* (undone from the database).
- A transaction begins with the first executable SQL statement. A transaction ends when it is committed or rolled back, either explicitly (with a COMMIT or ROLLBACK statement) or implicitly (when a DDL statement is issued).

Example of Transaction


Transaction Begins
Decrement Savings Account
Increment Checking Account
Record in Transaction Journal
End Transaction
Transaction Ends

```
UPDATE savings_accounts
  SET balance = balance - 500
  WHERE account = 3209;
```

```
UPDATE checking_accounts
  SET balance = balance + 500
  WHERE account = 3208;
```

```
INSERT INTO journal VALUES
  (journal_seq.NEXTVAL, '1B'
  3209, 3208, 500);
```

```
COMMIT WORK;
```

Before Commit

Before a transaction that has modified data is committed, the following will have occurred:

- Oracle has generated rollback segment records in rollback segment buffers of the SGA. The rollback information contains the old data values changed by the SQL statements of the transaction.
- Oracle has generated redo log entries in the redo log buffers of the SGA. These changes may go to disk before a transaction is committed.
- The changes have been made to the database buffers of the SGA. These changes may go to disk before a transaction actually is committed.

After Commit

After a transaction is committed, the following occurs:

- The internal transaction table for the associated rollback segment records that the transaction has committed, and the corresponding unique system change number (SCN) of the transaction is assigned and recorded in the table.
- LGWR writes the redo log entries in the redo log buffers of the SGA to the online redo log file. LGWR also writes the transaction's SCN to the online redo log file. This is the atomic event that constitutes the commit of the transaction.
- Oracle releases locks held on rows and tables.
- Oracle marks the transaction "complete".

Rolling Back

- *Rolling back* means undoing any changes to data that have been performed by SQL statements within an uncommitted transaction.
- Oracle allows you to roll back an entire uncommitted transaction. Alternatively, you can roll back the trailing portion of an uncommitted transaction to a marker called a savepoint.

Rolling Back (2)

In rolling back **an entire transaction**, without referencing any savepoints, the following occurs:

- Oracle undoes all changes made by all the SQL statements in the transaction by using the corresponding rollback segments.
- Oracle releases all the transaction's locks of data.
- The transaction ends.

In rolling back a transaction **to a savepoint**, the following occurs:

- Oracle rolls back only the statements executed after the savepoint.
- The specified savepoint is preserved, but all savepoints that were established after the specified one are lost.
- Oracle releases all table and row locks acquired since that savepoint, but retains all data locks acquired previous to the savepoint.
- The transaction remains active and can be continued.

Data Concurrency

- Many users can access data at the same time.
- Users should see a consistent view of the data, including visible changes made by the user's own transactions and transactions of other users.

Preventable Phenomena

- dirty reads: A transaction reads data that has been written by a transaction that has *not* been committed yet.
- non-repeatable (fuzzy) reads: A transaction re-reads data it has previously read and finds that another committed transaction has modified or deleted the data.
- phantom read: A transaction re-executes a query returning a set of rows that satisfy a search condition and finds that another committed transaction has inserted additional rows that satisfy the condition.

Isolation Levels

The SQL standard defines four levels of isolation in terms of the phenomena a transaction running at a particular isolation level is permitted to experience.

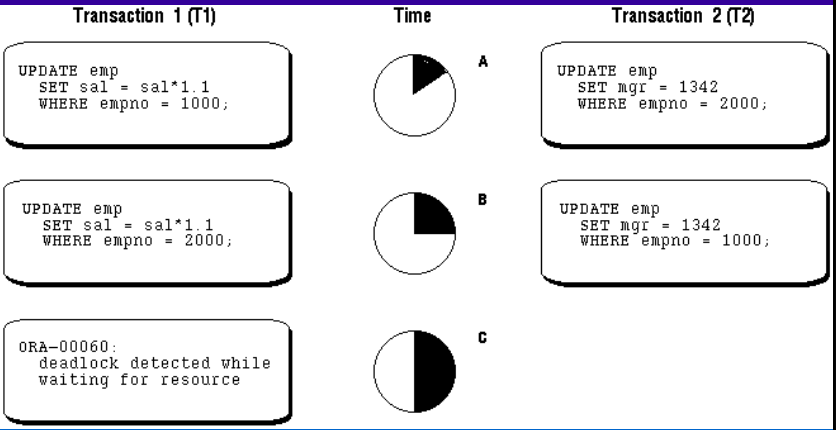
Isolation Level	**Dirty Read**	**Non-Repeatable Read**	**Phantom Read**
Read uncommitted	Possible	Possible	Possible
Read committed	Not possible	Possible	Possible
Repeatable read	Not possible	Not possible	Possible
Serializable	Not possible	Not possible	Not possible

Oracle offers the read committed and serializable isolation levels.

Locking Mechanisms

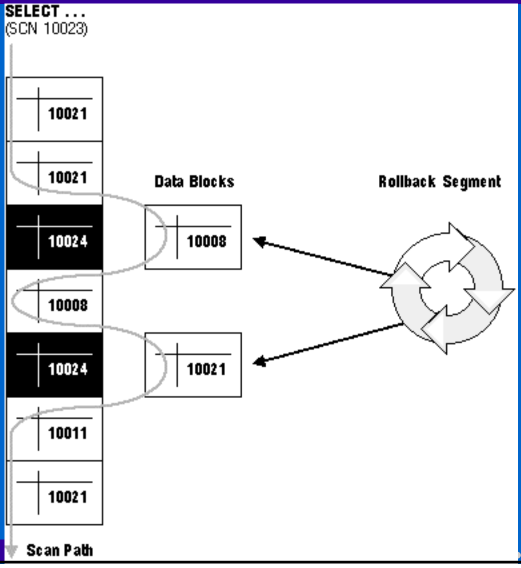
- *Locks* are mechanisms used to prevent destructive interaction between users accessing the same resource.
- Resources include two general types of objects:
  - user objects, such as tables and rows (structures and data)
  - system objects not visible to users, such as shared data structures in the memory and data dictionary rows
- In general, you can use two levels of locking in a multi-user database:
  - Exclusive locks
  - Shared locks

Deadlocks



Transaction 1 (T1)	Time	Transaction 2 (T2)
UPDATE emp SET sal = sal*1.1 WHERE empno = 1000;	A	UPDATE emp SET mgr = 1342 WHERE empno = 2000;
UPDATE emp SET sal = sal*1.1 WHERE empno = 2000;	B	UPDATE emp SET mgr = 1342 WHERE empno = 1000;
ORA-00060: deadlock detected while waiting for resource	C	

Multiversion Concurrency Control



Level of Consistency

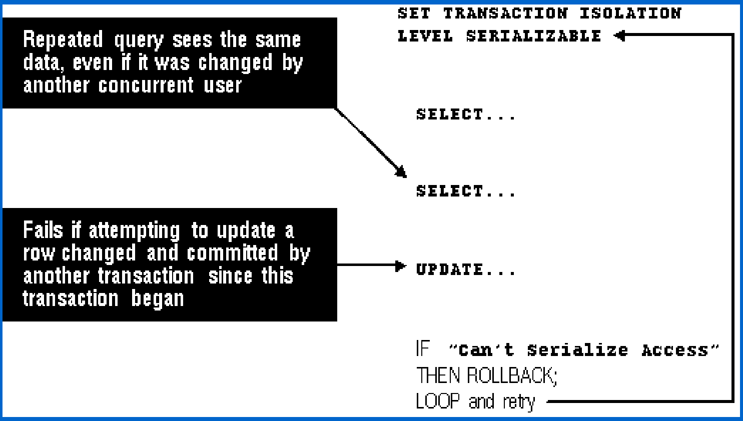
- Oracle always enforces *statement-level* read consistency. This guarantees that the data returned by a single query is consistent with respect to the time that the query began. Therefore, a query never sees dirty data nor any of the changes made by transactions that commit during query execution.
- Oracle also allows the option of enforcing *transaction-level read consistency*. When a transaction executes in serializable mode,all data accesses reflect the state of the database as of the time the transaction began. This means that the data seen by all queries within the same transaction is consistent with respect to a single point in time, except that queries made by a serializable transaction do see changes made by the transaction itself. Therefore, transaction-level read consistency produces repeatable reads and does not expose a query to phantoms.

Oracle Isolation Levels

Oracle provides three transaction isolation modes:

- *read committed*: This is the default transaction isolation level. Each query sees only data that was committed before the query (not the transaction) began. A transaction that executes a given query twice may experience both non-repeatable read and phantoms.
- *Serializable transactions* see only those changes that were committed at the time the transaction began, plus those changes made by the transaction itself.
- Read only: Read only transactions see only those changes that were committed at the time the transaction began and do not allow INSERT, UPDATE, and DELETE.

Problems with Serialization


Repeated query sees the same data, even if it was changed by another concurrent user
Fails if attempting to update a row changed and committed by another transaction since this transaction began
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE
SELECT...
SELECT...
UPDATE...
IF "Can't Serialize Access"
THEN ROLLBACK;
LOOP and retry

How Oracle Locks Data

- The only data locks Oracle acquires automatically are row-level locks.
- Oracle does not escalate locks from the row level to a coarser granularity.
- Readers of data do not wait for writers of the same data rows.
- Writers of data do not wait for readers of the same data rows (unless SELECT... FOR UPDATE is used, which specifically requests a lock for the reader).
- Writers only wait for other writers if they attempt to update the same rows at the same time.

How Oracle Locks Data (2)

- All locks acquired by statements within a transaction are held for the duration of the transaction.
- All locks acquired by statements within a transaction are held for the duration of the transaction.
- Oracle automatically detects deadlock situations and resolves them automatically by rolling back one of the statements involved in the deadlock, thereby releasing one set of the conflicting row locks.

Types of Locks

- Data (DML) locks protect data. For example, *table locks* lock entire tables, *row locks* lock selected rows.
- Dictionary (DDL) locks protect the structure of objects. For example, dictionary locks protect the definitions of tables and views.
- Internal locks and latches protect internal database structures such as datafiles. Internal locks and latches are entirely automatic.
- Parallel cache management locks are distributed locks that cover one or more data blocks (table or index blocks) in the buffer cache. PCM locks do not lock any rows on behalf of transactions.

Data Locks

- A transaction acquires an exclusive data lock (TX) for each individual row modified by one of the following statements: INSERT, UPDATE, DELETE, and SELECT with the FOR UPDATE clause.
- A modified row is **always** locked exclusively so that other users cannot modify the row until the transaction holding the lock is committed or rolled back.
- A transaction acquires a table lock (TM) when a table is modified in the following DML statements: INSERT, UPDATE, DELETE, SELECT with the FOR UPDATE clause, and LOCK TABLE.

Table Locks

SQL Statement	Mode of Table Lock	Lock Modes Permitted?				
		RS	RX	S	SRX	X
SELECT...FROM table ...	none	Y	Y	Y	Y	Y
INSERT INTO table ...	RX	Y	Y	N	N	N
UPDATE table ...	RX	Y*	Y*	N	N	N
DELETE FROM table ...	RX	Y*	Y*	N	N	N
SELECT ... FROM table FOR UPDATE OF ...	RS	Y*	Y*	Y*	Y*	N
LOCK TABLE table IN ROW SHARE MODE	RS	Y	Y	Y	Y	N
LOCK TABLE table IN SHARE MODE	RX	Y	Y	N	N	N
LOCK TABLE table IN SHARE MODE	S	Y	N	Y	N	N
LOCK TABLE table IN SHARE ROW EXCLUSIVE MODE	SRX	Y	N	N	N	N
LOCK TABLE table IN EXCLUSIVE MODE	X	N	N	N	N	N

Row Share (RS) and Row Exclusive (RX)

- High degree of concurrency
- Your transaction needs to prevent a table from being altered or dropped before the table can be modified later in your transaction

Share (S)

- Your transaction only queries the table and needs transaction-level read consistency on the table.

Share Row Exclusive (SRX)

- Your transaction requires both transaction-level read consistency and the ability to update the locked table.

Exclusive (X)

- Immediate update access.
- Transaction-level read consistency.

Locks Compatibility

	RS	RX	S	SRX	X
None	Y	Y	Y	Y	Y
RX	Y*	Y*	N	N	N
RS	Y*	Y*	Y*	Y*	N
S	Y	N	Y	N	N
SRX	Y	N	N	N	N
X	N	N	N	N	N

Example of explicit locking

Transaction 1	Time Point	Transaction 2

`LOCK TABLE scott.dept`
`  IN ROW SHARE MODE;`
`Statement processed`
	1	`DROP TABLE scott.dept;`
		`DROP TABLE scott.dept`
		`                *`
		`ORA-00054`
		`(exclusive DDL lock not possible because of T1's table lock)`
		`LOCK TABLE scott.dept`
		`   IN EXCLUSIVE MODE NOWAIT;`
		`ORA-00054`

	4	`SELECT LOC FROM scott.dept`
		`WHERE deptno = 20 FOR UPDATE`
		`OF loc; LOC - - - - - - -`
		`DALLAS 1 row selected`

`UPDATE scott.dept`
`  SET loc = 'NEW YORK'`
`WHERE deptno = 20;`
`(waits because T2 has locked same rows)`
	5	

	6	`ROLLBACK;` *(releases row locks)*

`1 row processed.`
`ROLLBACK;`
	7	

Part VI

Query Optimization

What is optimization?

- Whenever a DML statement is issued, Oracle must determine how to execute it.
- There may be different ways to execute the statement.
- The *optimizer*'s job is to choose one of these ways.

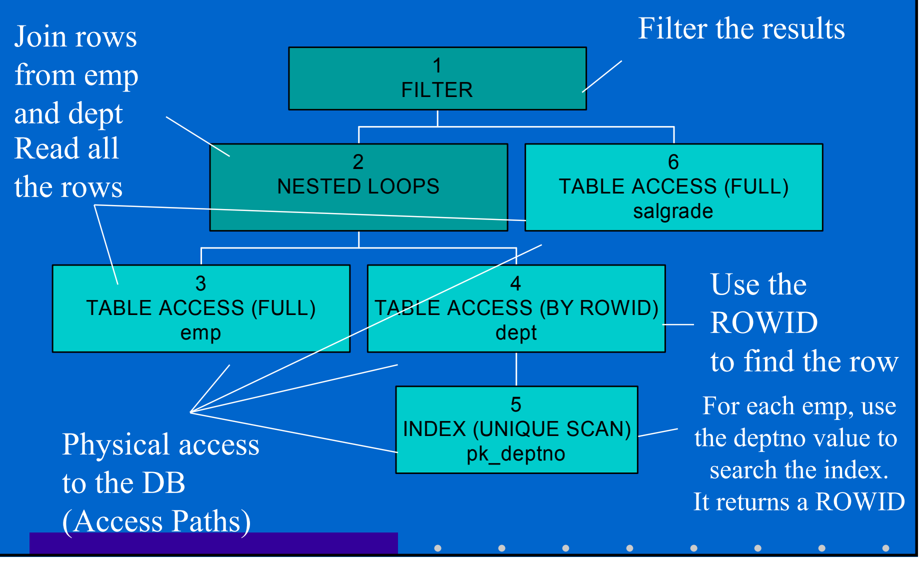
Execution Plans

- To execute a DML statement, ORACLE may have to perform many steps.
- Each step may:
  – retrieve rows from the DB.
  – prepare rows for the user in some way.

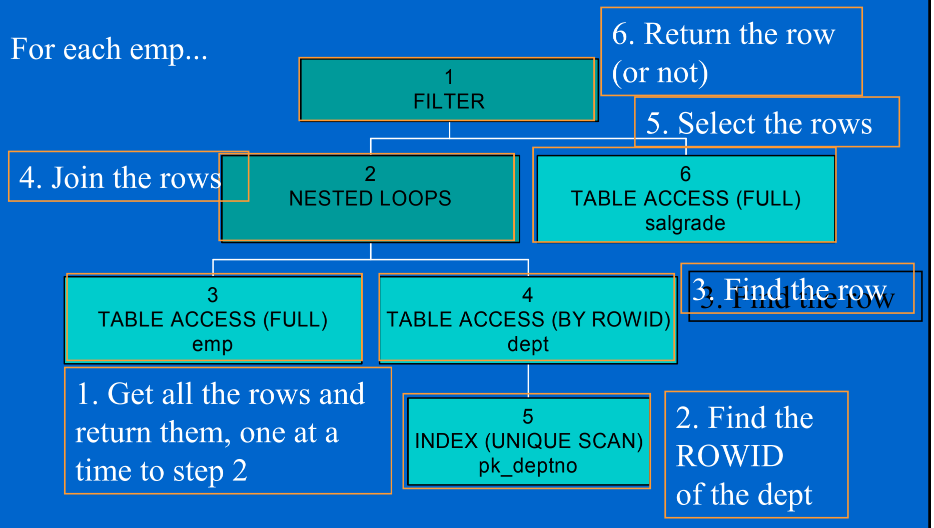
Example

```
SELECT ename,job,sal,dname
  FROM emp,dept
WHERE emp.deptno=dept.deptno
  AND NOT EXISTS
      (SELECT *
            FROM salgrade
            WHERE emp.sal BETWEEN
            lowsal AND hisal)
```

An Execution Plan


1
FILTER
2
NESTED LOOPS
6
TABLE ACCESS (FULL)
salgrade
3
TABLE ACCESS (FULL)
emp
4
TABLE ACCESS (BY ROWID)
dept
5
INDEX (UNIQUE SCAN)
pk_deptno
Filter the results
Join rows from emp and dept
Read all the rows
Use the ROWID to find the row
For each emp, use the deptno value to search the index. It returns a ROWID
Physical access to the DB (Access Paths)

Order of Execution



The explain plan command

ID	*Operation*	*Options*	*Object_name*
1	Filter		
2	Nested loops		
3	Table access	Full	EMP
4	Table access	By Rowid	DEPT
5	Index	Unique scan	pk_deptno
6	Filter	Full	SALGRADE

Two approaches to optimization

- Rule-based:

Choose an execution plan based on the access path available and choose the access path using a heuristic ranking.

- Cost-based
  - Generate a set of possible access paths.
  - Evaluate the cost of each access path based on the data distribution and statistics.
  - Choose the plan with the smallest cost.

How the optimization is done

- Evaluation of expression and conditions
- Statement transformation
- View merging
- Choice: rule-based or cost-based
- Choice of access paths
- Choice of join orders
- Choice of join operation

Types of SQL statements

- Simple statements
- Simple queries
- Joins
- Equijoins
- Nonequijoins
- Outerjoins
- Cartesian products
- Complex statements
- Compound query
- Statements accessing views

Evaluating Expressions and conditions

- Constants:
  - `sal > 32000/12`
  - `sal*12 > 1000`
- LIKE:
  - `ename LIKE 'SMITH'`
  - `ename = 'SMITH'`
- BETWEEN:
  - `sal BETWEEN 2000 AND 3000`
  - `sal >= 2000 AND sal <= 3000`

Evaluating Expressions and conditions (2)

Transitivity

```
Select *
From emp, dept
Where emp.deptno = 20 and
emp.deptno = dept.deptno;
```

replaced with

```
emp.deptno = 20;
```

Evaluating Expressions and conditions (3)

OR’s and Compound Queries

```
SELECT *
FROM EMP
WHERE job = ‘Clerk’
OR deptno = 10;
```

replaced with:

```
UNION ALL SELECT * FROM EMP
WHERE deptno = 10;
```

Optimizing Complex Statements

There are two approaches to the optimization of complex statements:

- Transform the complex statement in a join statement and optimize the join statement.
- Optimize the complex statement as it is.

Example of Transformation

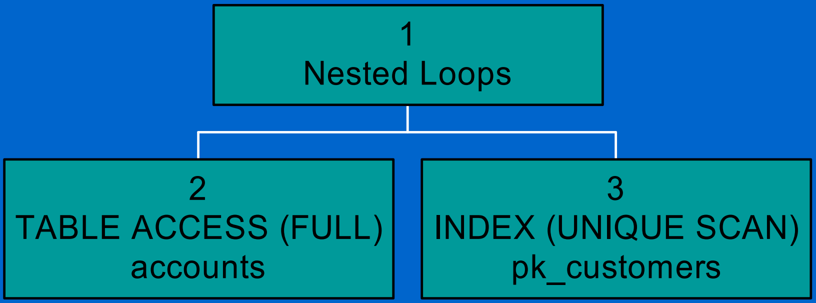
```
SELECT *
   FROM accounts
   WHERE custno IN (SELECT custno FROM
  customers);
```

is transformed into

```
SELECT accounts.*
 FROM accounts,customers
 WHERE account.custno =
     customers.custno
```

customer.custno must be a primary key

The corresponding execution plan



Complex Statements that are not simplified

- The query and its subqueries are optimized independently.
- For example:

```
SELECT *
FROM accounts
WHERE accounts.balance >
(SELECT AVG(balance) FROM
  accounts);
```

Statements that Access Views

There are three approaches to the optimization of queries that access views:

- Merge the view definition into the query and then optimize the resulting query.
- Merge the query in the view definition and then optimize the resulting query.
- Optimize and execute the view and then optimize and execute the query.

Merging the view definition

- View:

```
CREATE VIEW
  emp_10
AS SELECT *
FROM emp
WHERE deptno =
  10;
```

- Query:

```
SELECT empno
FROM emp_10
WHERE empno >
  7800
```

```
SELECT empno
FROM emp
WHERE empno>7800
and deptno = 10;
```

Merging the query (1)

- View:

```
CREATE VIEW emp
AS SELECT * FROM
  emp1
UNION SELECT *
  FROM emp2;
```

- Statement:

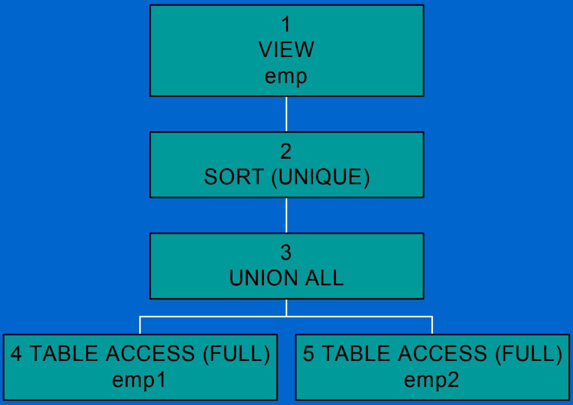
```
SELECT
  empno,ename
FROM emp
WHERE deptno=20;
```

Merging the query (2)

The query that is executed is

```
SELECT * FROM emp1 WHERE
  deptno=20
UNION
SELECT * FROM emp2 WHERE
  deptno=20
```

The execution plan



The query is merged in the view

If the view's query contains:

- Set operator
- Group by
- DISTINCT
- Group Function

Example

- View:

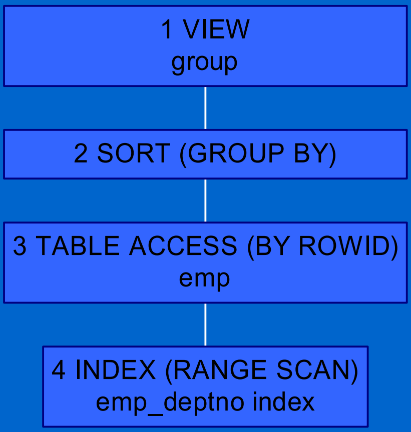
```
CREATE VIEW group
AS SELECT
   AVG(sal)
  avg_sal,
   MIN(sal)
  min_sal,
   MAX(sal) max_sal
FROM emp
GROUP BY deptno;
```

- Statement:

```
SELECT * FROM
group
WHERE deptno=10
```

```
Select
AVG(sal) avg_sal,
MIN(sal) min_sal,
MAX(sal) max_sal
FROM emp
WHERE deptno = 10 GROUP
 BY deptno
```

The execution plan



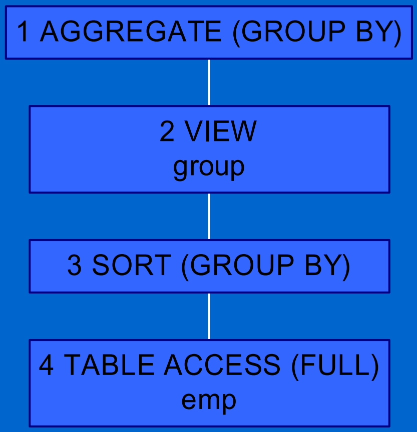
Example

- Statement:

```
Select
  AVG(avg_sal),AVG(min_sal),AVG(max_sal)
FROM group;
```

```
Select AVG(AVG(sal)),AVG(MIN(sal)),
AVG(MAX(sal))
FROM emp
GROUP by deptno
```

The execution plan



Optimizing other statements that access views

- ORACLE cannot always merge definitions and queries.
- In these cases, ORACLE issues the view’s query, collects the rows and then access this set of rows with the original statement as thought it was a table.

Example

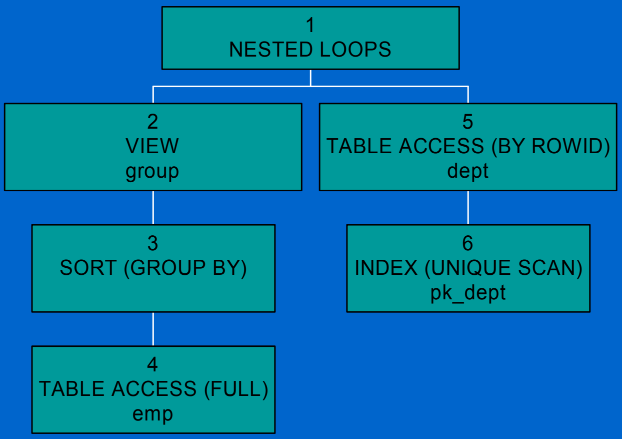
- View:

```
CREATE VIEW group
AS SELECT deptno,
AVG(sal) avg_sal,
MIN(sal) min_sal,
MAX(sal) max_sal
FROM emp
GROUP BY deptno
```

- Query:

```
SELECT
  group.deptno,
avg_sal, min_sal,
  max_sal,
dname,loc
FROM group,dept
WHERE
  group.deptno=dept
  .deptno
```

Execution Plan



Optimization Approach and Goal

- `OPTIMIZER_MODE`:
  - `COST`: If there are statistics then cost-based else rule-based.
  - `RULE`: rule-based approach.
- `ALTER SESSION`:
  - `CHOOSE, ALL_ROWS` (best throughput), `FIRST_ROWS` (best response time), RULE

Choosing Access Path

- The basic methods.
- The access paths and when they are available.
- How the optimizer chooses among the access paths.

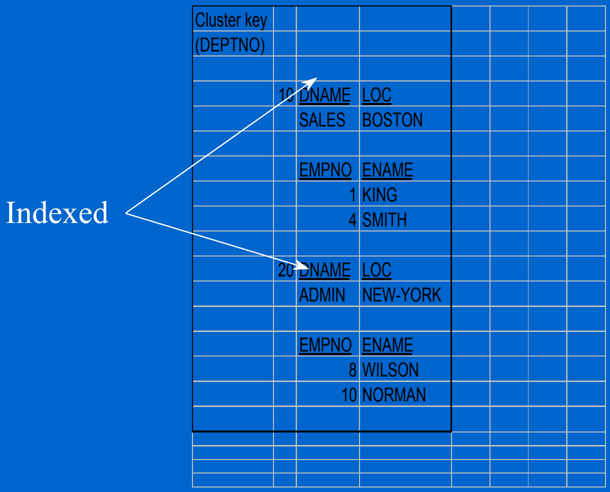
Access Methods

- Full Table Scans
  - Read each row and determine of it satisfies the statement's WHERE clause.
  - Implemented very efficiently using multi-block reads.
  - Each data block is read only once.
- Table Access by ROWID: fastest way.

Indexes, Clusters and Hash Clusters

- Indexes are created on one or more columns of a table. Very useful for range conditions. They are independent of the table.
- Clusters are an optional method for storing the table. Clusters group together tables because they share columns and often used together.
- The related columns in a cluster is the Cluster key (always indexed)

Example: Cluster



Cluster key (DEPTNO)		
10	DNAME	LOC
	SALES	BOSTON
	EMPNO	ENAME
	1	KING
	4	SMITH
20	DNAME	LOC
	ADMIN	NEW-YORK
	EMPNO	ENAME
	8	WILSON
	10	NORMAN

Hash Clusters

- Organization like simple clusters but...
- A row is stored in a hash cluster based on the result of applying a hash function to the row’s cluster key value.
- All rows with the same key value are stored together on the disk.

Example: Hash Cluster

Hash Key	Cluster Key	
	TRIALNO	Other Columns
237	1235	...
	2363	...
	7262	...
238	16262	...
	83747	...

Access Methods (2)

- Cluster Scans:
  - Retrieves all the rows that have the same cluster key value.
  - ORACLE first obtains the ROWID of one of the selected rows by scanning the cluster index.
  - ORACLE then locates the rows based on this ROWID.

Access Methods (3)

- Hash Scans:
  - ORACLE first obtains the hash value by applying the hash function.
  - ORACLE then scans the data blocks containing rows with that hash value.

Access Methods (4)

- Index Scans:
  - ORACLE searches the index for the indexed column values accessed by the statement.
  - If the statement accesses only columns of the index, ORACLE reads the values only from the index.
  - In the other case, ORACLE uses the ROWID found in the index to read the specific row from the table.

Access Methods (5)

- An index scan can be one of these types:
  - Unique: a unique scan of the index returns only a single ROWID.
  - Range: A range scan can return more than one ROWID.

1 - Single Row By ROWID

- If the *Where Clause* identifies the ROWID
- Example:

```
SELECT *
FROM emp
WHERE
  ROWID='00000DC5.0000.0001'
```

2 - Single Row by Cluster Join

- For statements that join tables stored in the same cluster if:
  - the statement equates every column of the cluster in each table.
  - there is a condition that guarantees that only one row will be returned.

Example

```
SELECT *
FROM emp,dept
WHERE emp.deptno =dept.deptno
AND emp.empno=7800
```

3 - Single Row by Hash Cluster Key

- The WHERE clause uses all columns of a hash cluster key with equality conditions.
- The statement is guaranteed to return only one row because the columns of the cluster key makes also a primary or unique key.

Example

```
SELECT *
FROM orders
WHERE orderno = 73468376
```

4 - Single Row by key

- The WHERE clause uses all the columns of a unique or primary key in equality condition.
- ORACLE uses the index of the key to find the ROWID and then accesses the row in the table.

Example

```
SELECT *
FROM emp
WHERE empno=7800
```

5 - Clustered Join

- The statement joins tables stored in the same cluster, that is the WHERE clause equates the columns of the cluster columns in the two tables.
- To execute the statement, ORACLE performs a nested loop operation.

Example

```
SELECT *
FROM emp,dept
WHERE emp.deptno = dept.deptno
```

6 - Hash Cluster Key

- The WHERE clause uses all the columns of the hash key in equality conditions.
- ORACLE calculates the hash value and then performs a hash scan.

```
SELECT *
FROM line_items
WHERE deptno=09870897
```

7 - Indexed Cluster Key

- ORACLE searches the cluster index to find the ROWID.
- ORACLE then scans the rows with the same cluster key using this ROWID.

```
SELECT *
FROM emp
WHERE deptno = 10
```

8 - Composite Index

- All the columns of the index are in the WHERE clause in equality conditions.
- ORACLE scans the index for the ROWIDs and then accesses the table

```
SELECT *
FROM emp
WHERE job='CLERK' AND deptno=30
```

9 - Single Column indexes

```
SELECT * FROM emp WHERE
  job='CLERK'
```

- ORACLE can merge indexes if the query conditions uses columns of many single column indexes

10 - Bounded Range Search on Indexed Columns

- The conditions uses either a single-column index or the leading portion of a composite index:

```
column = expr
column >[=] expr AND column <[=] expr
column BETWEEN expr AND expr
column LIKE ‘c%’
```

- ORACLE performs a range scan on the index and then accesses the table by ROWID.

11 - Unbounded Range Search on Indexed Columns

```
column >[=] expr
column <[=] expr
```

12 - Sort Merge Join

- Join on non-clustered columns.

13 - Max or Min of Indexed Column

```
SELECT MAX(sal)
FROM emp
```

- ORACLE performs a range scan on the index

14 - ORDER BY on indexed column

```
SELECT *
FROM emp
ORDER BY empno
```

15 - Full Table Scan

- For any SQL statement.
- Remark: You cannot use a index if you have

```
column1 (>|<|>=|<=) column2
column IS NULL
column IS NOT NULL
column NOT IN
column !=
column LIKE
```

column1 and column2 are in the same table.

Rule-based optimization

SELECT empno
FROM emp
WHERE ename = 'CHUNG'
AND sal > 2000

Indexed

Index

Single-Column Index (RANK 9)
Unbounded range index scan (Rank 11)
Full table scan (Rank 15)

Cost-Based Optimization

To choose among the available access paths, the optimizer considers these factors:

- Selectivity of the query. (big % : Full scan, small %: index scan).
- `DB_FILE_MULTIBLOCK_READ_COUNT` (high: Full scan, small: index scan)

Example of cost-based opt.

```
SELECT * FROM emp WHERE ename = 'JACKSON'
```

- If ename is a unique or primary key, the optimizer determines that this query is highly selective and that it must use the index.
- If ename is not a key, the optimizer uses the following statistical values:

`USER_TAB_COLUMNS.NUM_DISTINCT`

`USER_TABLES.NUM_ROWS`

By assuming that the ename values are uniformly distributed, the optimizer can evaluate the selectivity

Example

```
SELECT * FROM emp WHERE empno < 7500
```

To estimate the selectivity of this query, the optimizer uses the boundary value of 7500 the statistics `HIGH_VALUE` and `LOW_VALUE` in the `USER_TAB_COLUMNS` statistics table. It assumes that the values are evenly distributed in the range to estimate the selectivity of the condition.

Example

```
SELECT * FROM emp WHERE empno < :e1
```

The value of the bind variable `:e1` is unknown at optimization time, therefore the optimizer heuristically guesses 25% of selectivity.

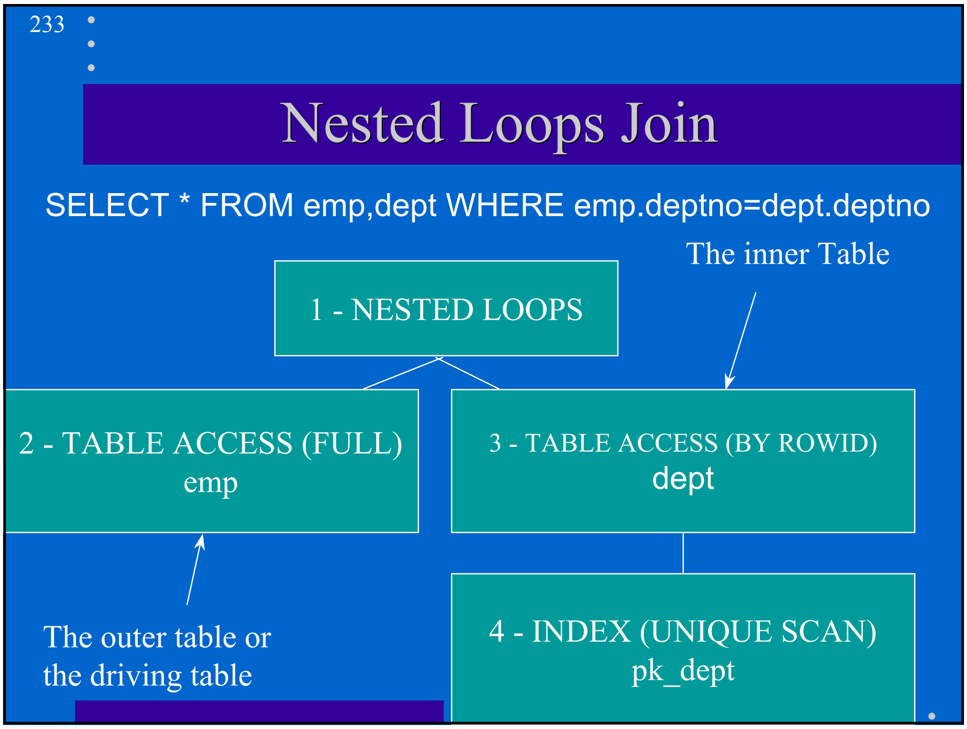
Optimizing Join Statements

To choose an execution plan for a join statement, the optimizer must choose:

- Access Paths
- Join Operation
- Join Order

These Choices are related.

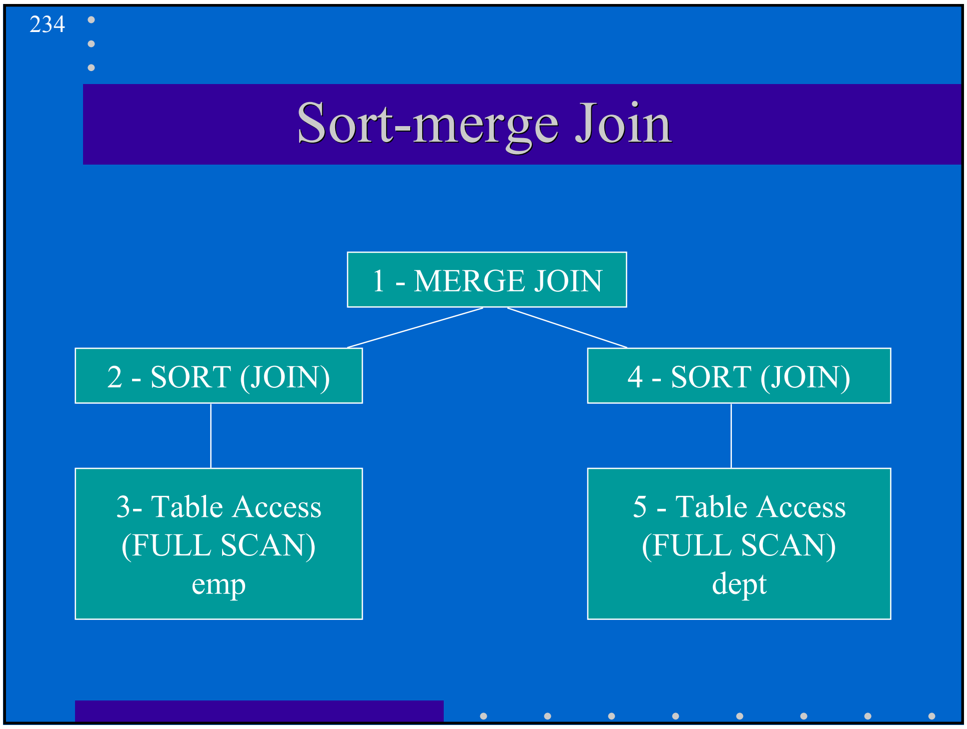
Lets examine the different possible join operations.



Nested Loops Join

SELECT * FROM emp,dept WHERE emp.deptno=dept.deptno

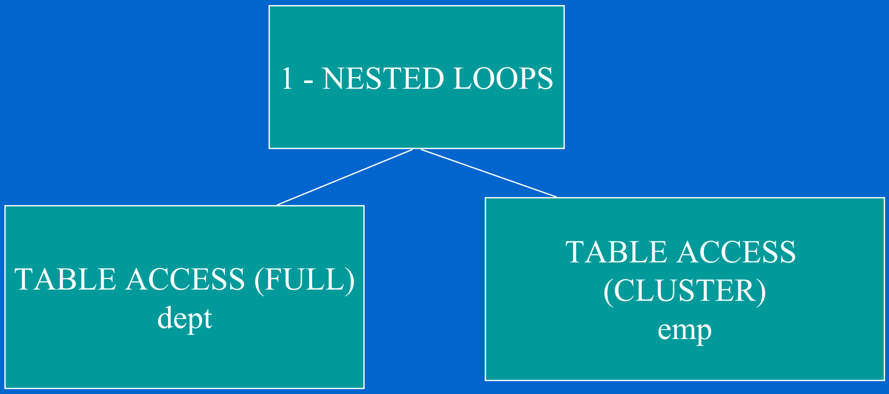
The inner Table
1 - NESTED LOOPS
2 - TABLE ACCESS (FULL) emp
3 - TABLE ACCESS (BY ROWID) dept
4 - INDEX (UNIQUE SCAN) pk_dept
The outer table or the driving table



Sort-merge Join

1 - MERGE JOIN
2 - SORT (JOIN)
4 - SORT (JOIN)
3- Table Access (FULL SCAN) emp
5 - Table Access (FULL SCAN) dept

Cluster Join



Choosing execution plans for joins

- Rule-based vs. Cost-based
- In the two approaches, the optimizers always:
  - begins with the joins that results in a single row (using keys).
  - begin with the tables without the outer join operators (+)

Rule-based Approach

- The optimizer generates a set of *R* join orders by following the algorithm:
  - Begin with a different table each time.
  - Choose an access path for each table.
  - Order the tables in the join from high ranked access path to lower rank access path.
  - For each table in the join, the optimizer must decide how to join the table to the previous results.

How to join the tables in the join

- If the access path is ranked 11 or better (there is an index or a cluster), use nested loops.
- If this is an equijoin, use sort-merge.
- Use nested loops.

How to decide which join to execute

The goal: maximize the number of nested loops join where there is an index on the inner table.

This is done using the following algorithm:

The Algorithm

Choose the table with:

- fewest nested loops where the inner table is access with full table scan.
- If there is a tie, choose the fewest sort-merge operation.
- If there is a tie, choose the plan with the highest rank for the first table:

The Algorithm (cont)

- If there is a tie between tables accessed by single-column index, choose the plan whose first table is accessed with most merged indexes.
- If there is a tie between tables accessed by bounded range scan, the optimizer chooses a plan whose first table is accessed with the greatest number of leading columns.

Cost-based optimization

Generate a set of join orders, join operations and access paths and estimate the cost of each one.

The optimizer estimates the cost using the following algorithm:

The Algorithm (cost-based)

- The cost of nested loops operation is based on the cost of reading each of the selected row of the outer table and each of its matching rows of the inner table in memory.
- The cost of a sort-merge join is mostly the cost of reading the sources in memory and sorting them

The Algorithm (cost-based)

- The influencing factors are:
  - SORT_AREA_SIZE
  - DB_FILE_MULTIBLOCK_READ_COUNT

Compound queries

- Execute each query separately and then apply the operator used in the compound query.

```
SELECT part FROM orders1
UNION ALL
SELECT part FROM orders2
```

Part VII

Security

Database Security

- **D**atabase security involves allowing or disallowing users from performing actions on the database and the objects within it.
- A privilege is permission to access a named object in a prescribed manner; for example, permission to query a table.
- Because privileges are granted to users at the discretion of other users, this is called *discretionary security*.

Users

- A *user* (sometimes called a *username*) is a name defined in the database that can connect to and access objects in database schemas.
- To access a database, a user must run a database application (such as an Oracle Forms form, SQL*Plus, or a Precompiler program) and connect using a username defined in the database.
- When a database user is created, a corresponding schema of the same name is created for the user.

Authentication

- Oracle provides user validation via three different methods for normal database users:
  - authentication by the operating system
  - authentication by a network authentication service
  - authentication by the associated Oracle database

User Tablespace Settings

the database administrator can set several options regarding tablespace usage:

- the user's default tablespace
- the user's temporary tablespace
- space usage quotas on tablespaces of the database for the user

PUBLIC

- Each database contains a user group called PUBLIC.
- The PUBLIC user group provides public access to specific schema objects (tables, views, and so on) and provides all users with specific system privileges.
- Every user automatically belongs to the PUBLIC user group.

User Resources

- As part of a user's security domain, the DBA can set limits on the amount of various system resources available to the user.
- Session level, Call level
- Ressources CPU Time, Logical Reads

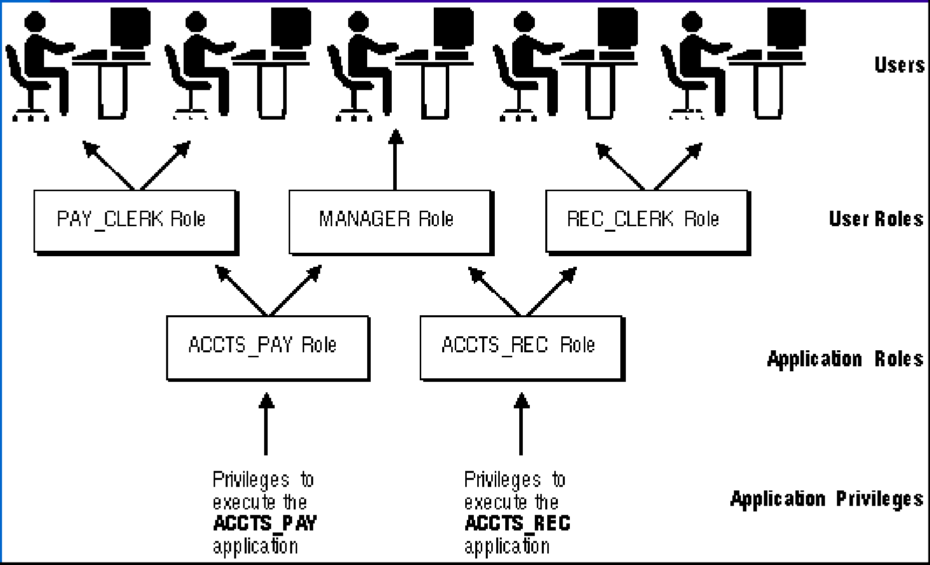
Privileges

- A *privilege* is a right to execute a particular type of SQL statement or to access another user's object. Some examples of privileges include
  - the right to connect to the database (create a session)
  - the right to create a table
  - the right to select rows from another user's table
  - the right to execute another user's stored procedure

Granting Privileges

- A user can receive a privilege in two different ways:
  - The DBA can grant privileges to users explicitly. For example, she can explicitly grant the privilege to insert records into the EMP table to the user SCOTT.
  - The DBA can also grant privileges to a role (a named group of privileges), and then grant the role to one or more users. For example, she can grant the privileges to select, insert, update, and delete records from the EMP table to the role named CLERK, which in turn you can grant to the users SCOTT and BRIAN.

Roles



Auditing

- *Auditing* is the monitoring and recording of selected user database actions. Auditing is normally used to
  - investigate suspicious activity. For example, if an unauthorized user is deleting data from tables, the security administrator might decide to audit all connections to the database and all successful and unsuccessful deletions of rows from all tables in the database.
  - monitor and gather data about specific database activities. For example, the database administrator can gather statistics about which tables are being updated, how many logical I/Os are performed, or how many concurrent users connect at peak times.

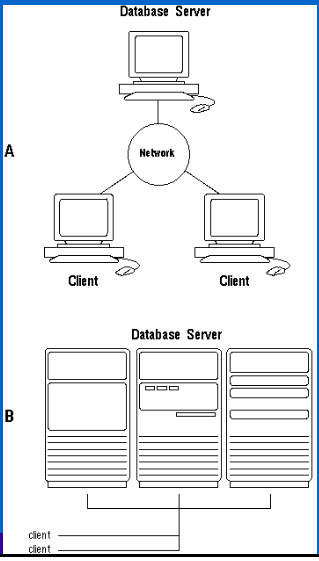
Part VIII

Distributed Processing
and
Distributed Databases

Client/Server

- In the Oracle client/server architecture, the database application and the database are separated into two parts: a front-end or client portion, and a back-end or server portion.
- The client executes the database application that accesses database information and interacts with a user.
- The server executes the Oracle software and handles the functions required for concurrent, shared data access to an Oracle database.

Distributed Processing



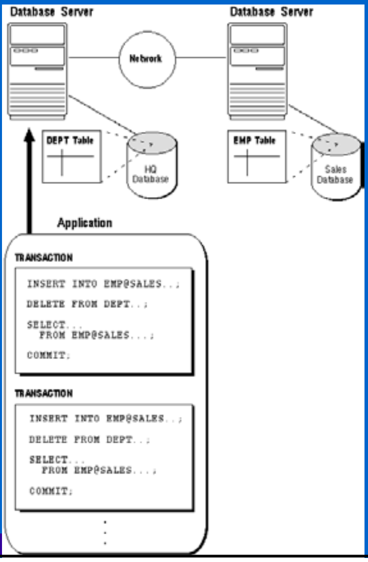
Advantages

- Client applications are not responsible for performing any data processing
- Client applications can be designed with no dependence on the physical location of the data
- Oracle can be *scaled*
- Easier and more efficient to manage concurrent access.
- Inexpensive, low-end client workstations can be used
- Network traffic is kept to a minimum because only the requests and the results are shipped over the network.

Distributed Databases

- A *distributed database* appears to a user as a single database but is, in fact, a set of databases stored on multiple computers.
- The data on several computers can be simultaneously accessed and modified using a network.
- Each database server in the distributed database is controlled by its local DBMS, and each cooperates to maintain the consistency of the global database

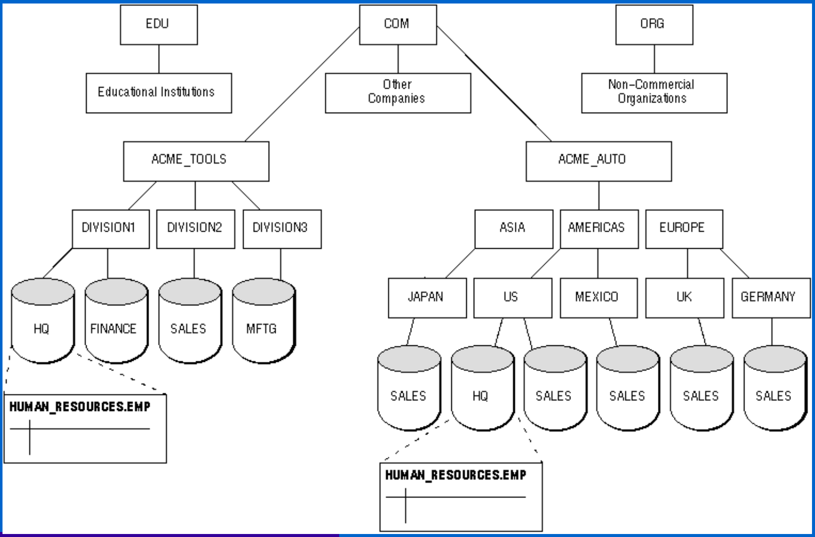
A Distributed Database



```
INSERT INTO EMP@SALES..;
DELETE FROM DEPT..;
SELECT...
  FROM EMP@SALES...;
COMMIT;
```

```
INSERT INTO EMP@SALES..;
DELETE FROM DEPT..;
SELECT...
  FROM EMP@SALES...;
COMMIT;
```

Schema Objects and Naming



Statements and Transactions

- A *remote query* is a query that selects information from one or more remote tables, all of which reside at the same remote node.
- A *remote update* is an update that modifies data in one or more tables, all of which are located at the same remote node.
- A *distributed query* retrieves information from two or more nodes.
- A *distributed update* modifies data on two or more nodes.
- A *remote transaction* is a transaction that contains one or more remote statements, all of which reference the same remote node.
- A *distributed transaction* is any transaction that includes one or more statements that, individually or as a group, update data on two or more distinct nodes of a distributed database.

Distributed Transaction

- All participants (nodes) in a distributed transaction should be unanimous as to the action to take on that transaction. That is, they should either all commit or rollback.
- Oracle automatically controls and monitors the commit or rollback of a distributed transaction and maintains the integrity of the global database using a transaction management mechanism known as *two-phase commit*.
- This mechanism is completely transparent. Its use requires no programming on the part of the user or application developer.

Prepare and Commit

The committing a distributed transaction has two distinct phases:

- Prepare: The global coordinator (initiating node) asks participants to prepare (to promise to commit or rollback the transaction, even if there is a failure).
- Commit: If all participants respond to the coordinator that they are prepared, the coordinator asks all nodes to commit the transaction. If any participants cannot prepare, the coordinator asks all nodes to roll back the transaction.

Prepare Phase

- By preparing, a node records enough information so that it can subsequently either commit or abort the transaction regardless of intervening failures.
- When a node is told to prepare, it can respond with one of three responses:
  - Prepared: Data on the node has been modified by a statement in the distributed transaction, and the node has successfully prepared.
  - Read-only: No data on the node has been, or can be, modified (only queried), so no prepare is necessary.
  - Abort: The node cannot successfully prepare.

Prepare Actions

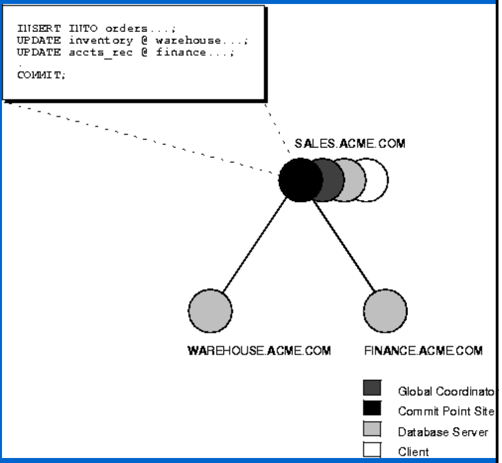
- The node requests its descendants to prepare.
- The node checks to see if the transaction changes data on that node or any of its descendants. If there is no change, the node skips the next steps and replies with a read-only message
- The node allocates all resources it needs to commit the transaction if data is changed.
- The node flushes any entries corresponding to changes made by that transaction to its local redo log.
- The node responds to the node that referenced it in the distributed transaction with a prepared message or, if its prepare or the prepare of one of its descendents was unsuccessful, with an abort message.

Commit Phase

- Before this phase occurs, all nodes referenced in the distributed transaction have guaranteed that they have the necessary resources to commit the transaction. That is, they are all prepared.
- Therefore, the commit phase consists of the following steps:
  - The global coordinator send a message to all nodes telling them to commit the transaction.
  - At each node, Oracle commits the local portion of the distributed transaction (releasing locks) and records an additional redo entry in the local redo log, indicating that the transaction has committed.
- When the commit phase is complete, the data on all nodes of the distributed system are consistent with one another.

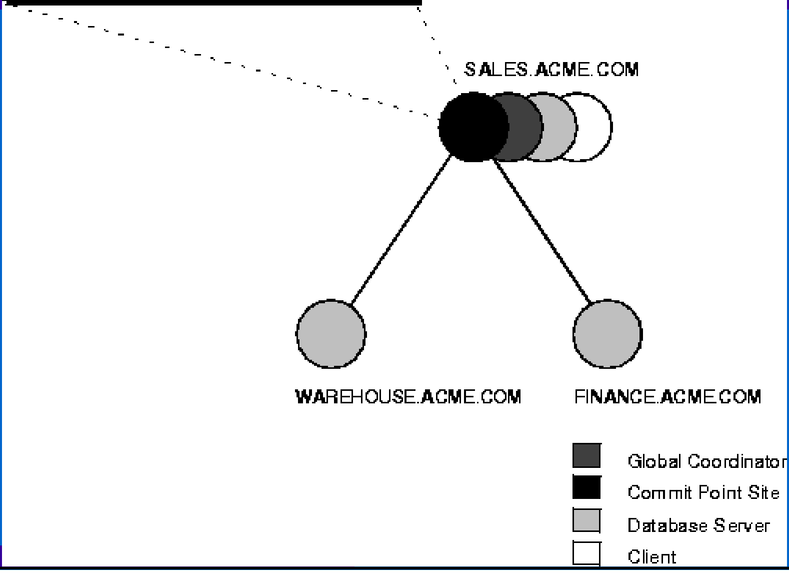
The Session Tree

- All nodes participating in the session tree of a distributed transaction assume one or more roles:
  - a client
  - a database server
  - a global coordinator
  - a local coordinator
  - the commit point site



```
INSERT INTO orders...;
UPDATE inventory @ warehouse...;
UPDATE accts_rec @ finance...;
.
COMMIT;
```

```
INSERT INTO orders...;
UPDATE inventory @ warehouse...;
UPDATE accts_rec @ finance...;
.
COMMIT;
```



■	Global Coordinator
■	Commit Point Site
▨	Database Server
□	Client

Clients and Servers

- A *client* references information from another node's database.
- A *server* is a node that is directly referenced in a distributed transaction or is requested to participate in a transaction because another node requires data from its database. A node supporting a database is also called a *database server*.

Local Coordinator

- A node that must reference data on other nodes to complete its part in the distributed transaction is called a *local coordinator*.
- A local coordinator is responsible for coordinating the transaction among the nodes it communicates directly with by:
  - receiving and relaying transaction status information to and from those nodes
  - passing queries to those node
  - receiving queries from those nodes and passing them on to other nodes
  - returning the results of queries to the nodes that initiated them

Global Coordinator

- The node where the distributed transaction originates (to which the database application issuing the distributed transaction is directly connected) is called the global coordinator.
- All of the distributed transaction's SQL statements, remote procedure calls, etc. are sent by the global coordinator to the directly referenced nodes, thus forming the session tree.

The Commit Point Site

- The job of the commit point site is to initiate a commit or roll back as instructed by the global coordinator.
- The system administrator always designates one node to be the commit point site in the session tree by assigning all nodes a commit point strength.
- The node selected as commit point site should be that node that stores the most critical data (the data most widely used)

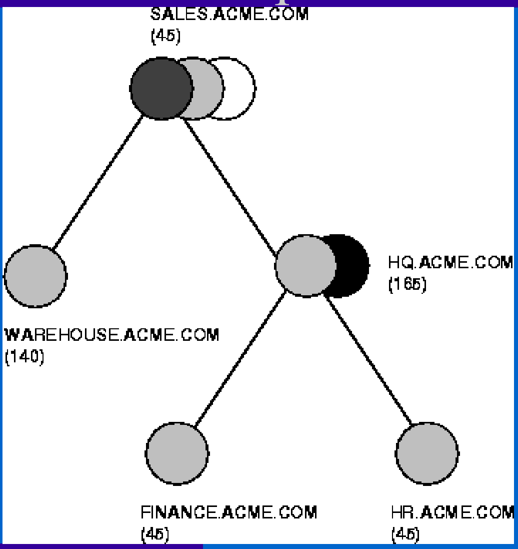
The Commit Point Site (2)

- The commit point site is distinct from all other nodes involved in a distributed transaction with respect to the following two issues:
    - The commit point site never enters the prepared state. This is potentially advantageous because if the commit point site stores the most critical data, this data never remains in-doubt, even if a failure situation occurs.
    - In effect, the outcome of a distributed transaction at the commit point site determines whether the transaction at all nodes is committed or rolled back.

Commit Point Strength

- Every node acting as a database server must be assigned a commit point strength.
- If a database server is referenced in a distributed transaction, the value of its commit point strength determines what role it plays in the two-phase commit.
- Specifically, the commit point strength determines whether a given node is the commit point site in the distributed transaction.

Example



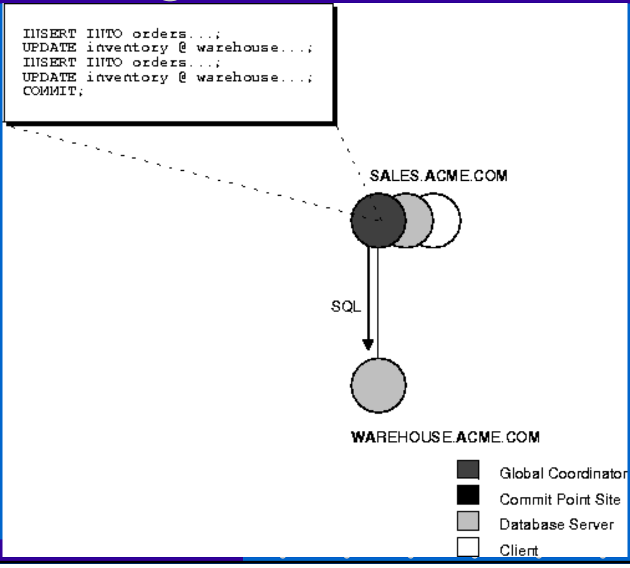
Selecting the Commit Point Site

- The commit point site is selected only from the nodes participating in the transaction.
- Once it has been determined, the global coordinator sends prepare messages to all participating nodes.
- Of the nodes directly referenced by the global coordinator, the node with the highest commit point strength is selected.

A Scenario

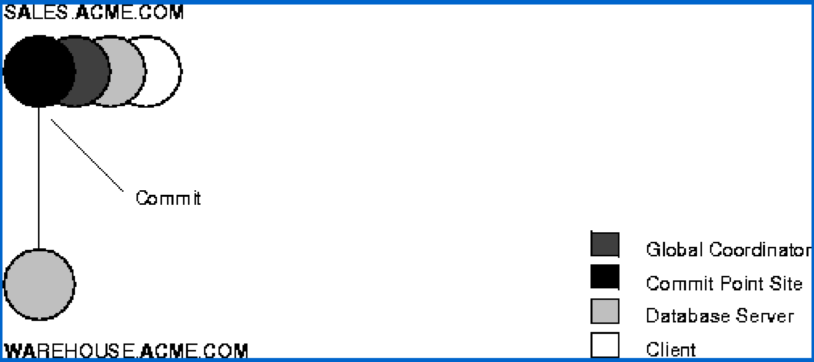
- A company that has separate Oracle servers, SALES.ACME.COM and WAREHOUSE.ACME.COM.
- As sales records are inserted into the SALES database, associated records are being updated at the WAREHOUSE database.

Defining the Session Tree

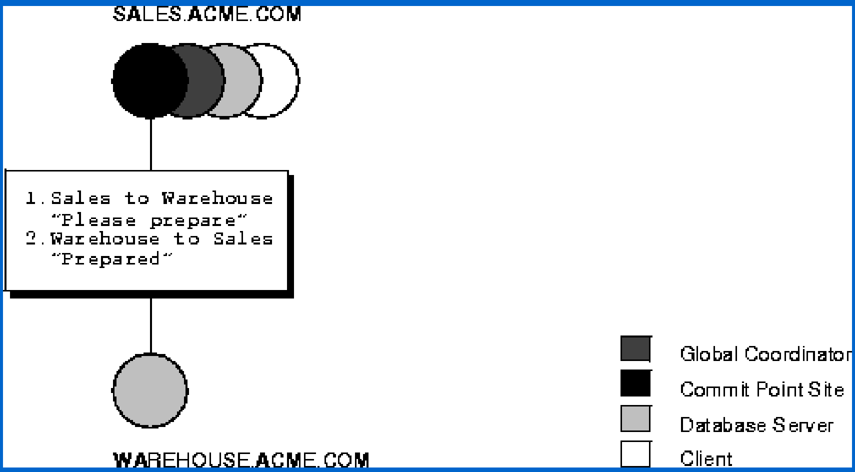


```
INSERT INTO orders...;
UPDATE inventory @ warehouse...;
INSERT INTO orders...;
UPDATE inventory @ warehouse...;
COMMIT;
```

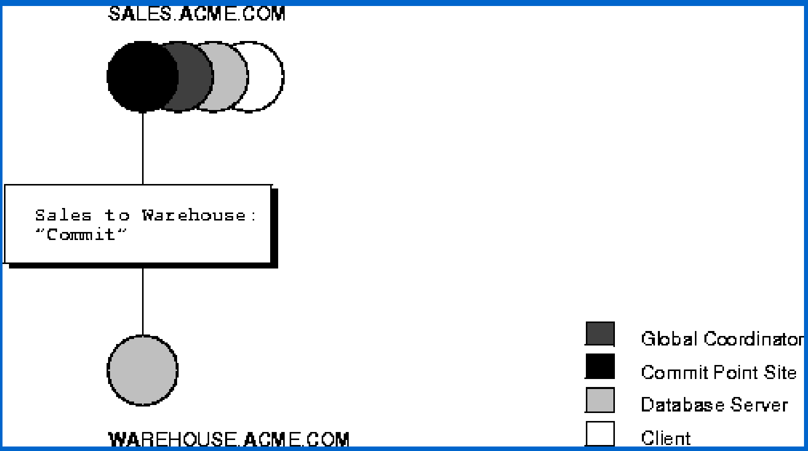
Determining the Commit Point Site



Prepare



Commit



Data Replication

Data Replication

- Data replication is the capability of maintaining copies of tables/databases separate from the primary copy.
- This replication is usually performed separately from a two phase commit protocol.
- This term is used by different vendors (in our case: Sybase, ORACLE, Informix) to describe differing functionality.

Definitions

- The Sybase *replication server* is a method for distributing copies of a table to multiple sites.
- The Oracle *table snapshot* is a method for distributing *read-only* copies of a table to multiple sites.
- The Informix *data replication feature* is a method for maintaining an exact duplicate of an Online instance elsewhere on the system.

Sybase Replication Server

- Designed to dynamically maintain subsets of data in a distributed database environment.
- A replication server environment is usually made up of LANs connected to WANs. Each LAN may have one or more replication server.

Primary Sites

- Each piece of data has a primary site. This can be thought as the original copy of the data.
- The primary site determines who can replicate the data and what data can be replicated.

Subscriptions

- Replication is started by the remote sites requesting data from the primary site.
- This is called *subscription*.
- Subscriptions look like SQL SELECT statement.

Subscriptions

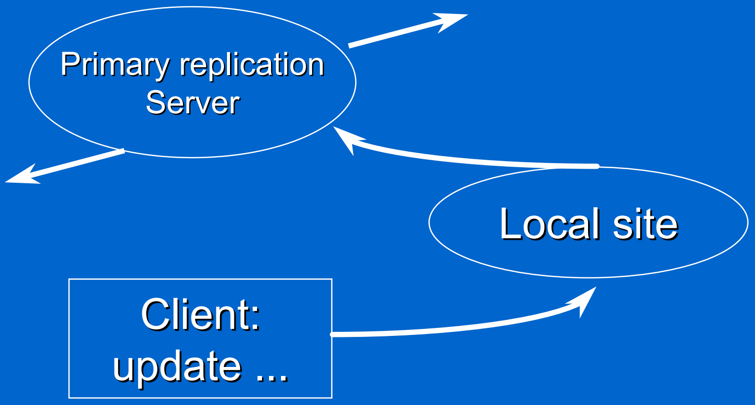
- After a subscription is entered, the primary site copies all the data that satisfies the subscription query to the remote site.
- From that point on, the remote site is kept up-to-date using one of the three methods described below.

Subscriptions (2)

- Sybase supports the change of the subscription criteria. This is called dynamic subscriptions.
- In case that any replication server fails, the DB stores all the transactions for that server and automatically resynchronizes the server once it has been recovered.

Replication methods (1)

Local-first Update



Drawbacks

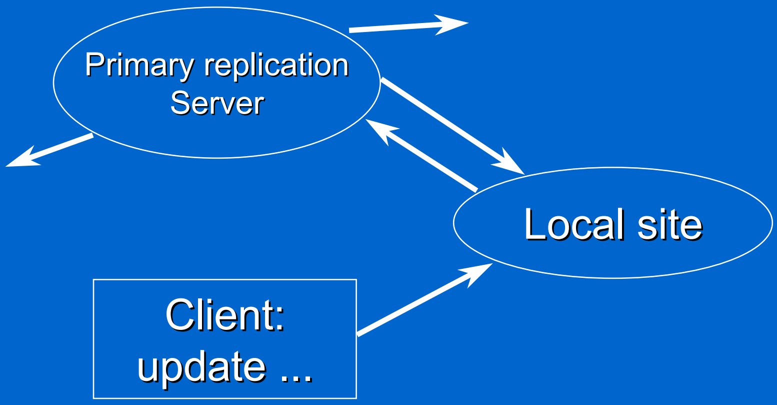
- The primary site may be down when the remote transaction is committed.
- Therefore, other sites will not work with up-to-date information.

Benefits

- The local site contains up-to-date information and may continue to proceed even if the replication server has failed.

Replication methods (2)

Primary-first Update



Drawbacks

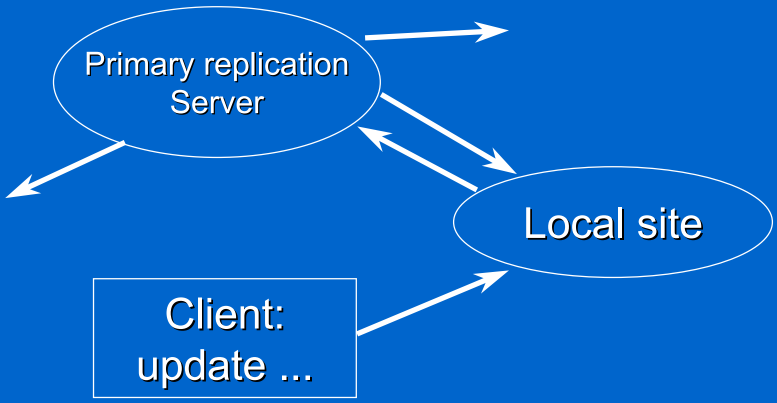
- This method depends upon the primary replication server being available.

Benefits

- Ensures that, if the replication server does not fail, users have a consistent view of the data.

Replication methods (3)

Version-Controlled Update



Drawbacks

- Applications are not aware that they are working with out-of-data information until commit.

Benefits

- ?

Oracle

- The Oracle snapshot is the result of a query of tables, views or other snapshots.
- Oracle defines two types of snapshots:
  - Simple snapshots: on a single table without GROUP BY or CONNECT BY.
  - complex snapshots
- The original tables are called the *master tables*. The snapshot is used for read-only access.

Refreshing the snapshot

- ORACLE supports two methods.
- The method depends on whether the snapshot is simple or complex and how often the user wants the information to be updated at the remote site.

Fast refresh

- Available only for simple snapshots.
- The snapshot log is a table found in the master DB used to track all the updates to the master table.
- A fast refresh applies the changes saved in the snapshot log to the snapshot periodically.

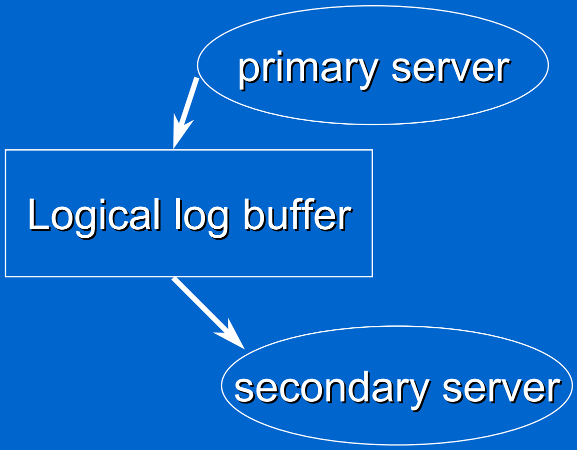
Complete refresh

- Complete refresh entirely rebuild the snapshot periodically.
- For the two refreshing methods, refreshing can be set manually or automatically.

Informix-Online Dynamic Server

- The aim: to maintain available copy of critical DB.
- The administrator sets a primary and a secondary DB instance.
- The secondary instance is read-only if the primary server is on-line.
- If the primary server fails, the secondary server may be used for read-write access.

Online data replication



Synchronization

- The servers synchronization is set up using two parameters:
  - DRINTERVAL: determines the interval between two resynchronizations of the servers.
  - DRTIMEOUT: how long the primary site waits for acknowledgment from the secondary site after the data has been transferred.