DATABASE CONCEPTS

Oracle Memory Structures

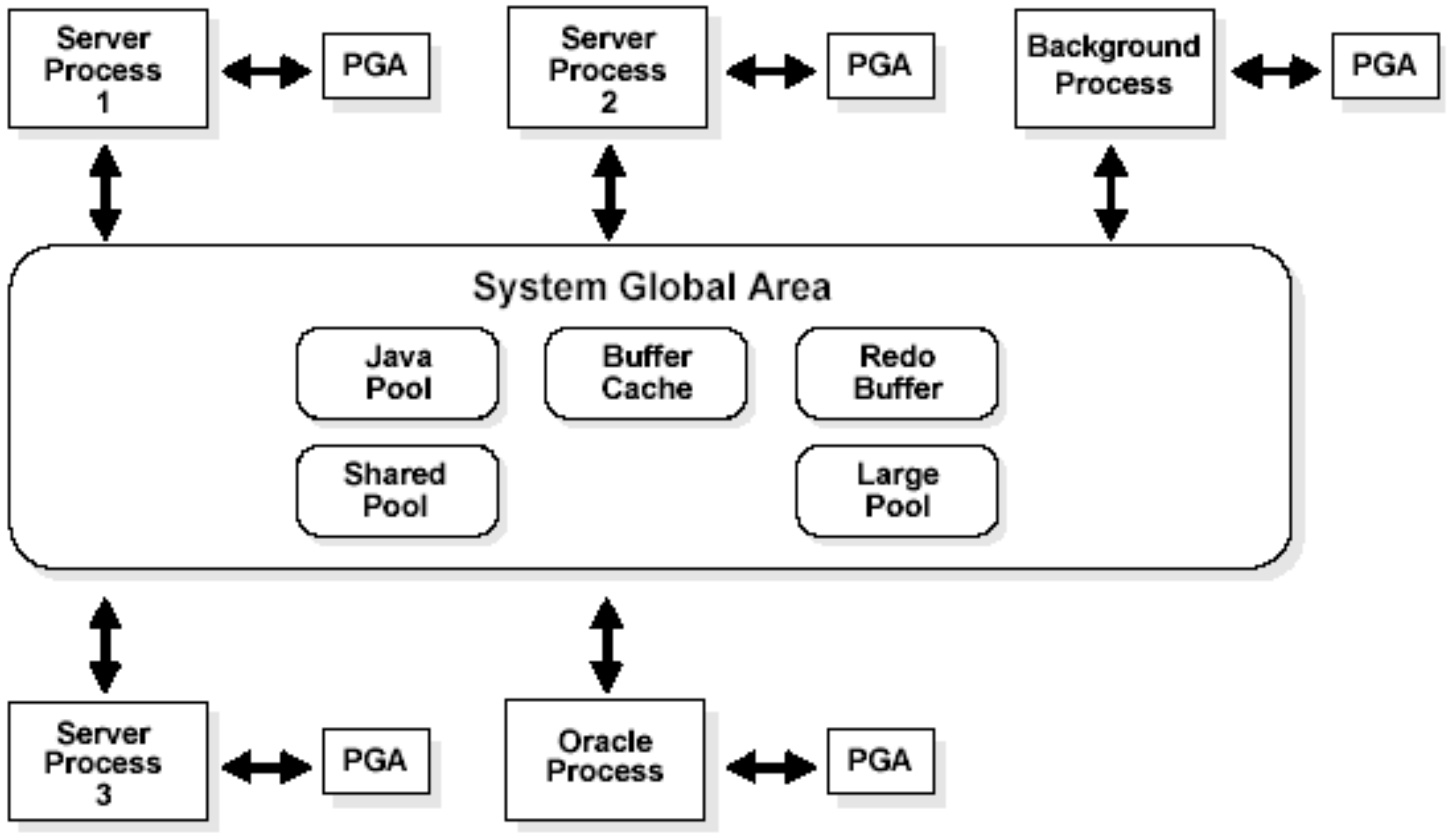
- Oracle uses memory to store information such as
  - Program code
  - Connected session information
  - Information needed during program execution
  - Information that is shared and communicated
  - Cached data

- The basic memory structures associated with Oracle include:
  - System Global Area (SGA)
  - Program Global Areas (PGA)

- System Global Area (SGA) is shared by all server and background processes and holds the following:
  - Redo log buffer
  - Database buffer cache
  - Shared pool
  - Large pool (if configured)

- Program Global Areas (PGA), which is private to each server and background process and holds the following:
  - Stack areas
  - Data areas

Oracle Memory Structures



System Global Area (SGA)

- A **system global area** (SGA) is a group of shared memory structures that contain data and control information for one Oracle database instance.
- If multiple users are concurrently connected to the same instance, then the data in the instance’s SGA is shared among the users. Consequently, the SGA is sometimes called the **shared global area**.
- An SGA and Oracle processes constitute an Oracle instance.

SGA Contd…

- Oracle automatically allocates memory for an SGA when you start an instance, and the operating system reclaims the memory when you shut down the instance.
- The SGA is read/write. All users connected to a multiple-process database instance can read information contained within the instance’s SGA, and several processes write to the SGA during execution of Oracle.

SGA Contd…

- The SGA contains the following data structures:
  - Database buffer cache
  - Redo log buffer
  - Shared pool
  - Java pool
  - Large pool (optional)
  - Data dictionary cache
  - Other miscellaneous information
- Part of the SGA contains general information about the state of the database and the instance, which the background processes need to access; this is called the **fixed SGA**.

SGA Contd…

- No user data is stored here.
- The SGA also includes information communicated between processes, such as locking information.
- If the system uses shared server architecture, then the request and response queues and some contents of the PGA are in the SGA.

SGA Contd..

- Oracle can change its SGA configuration while the instance is running. With the ***dynamic SGA*** infrastructure, the sizes of the buffer cache, the shared pool, and the large pool can be changed without shutting down the instance.
- Dynamic SGA also allows Oracle to set, at run time, limits on how much virtual memory Oracle will use for the SGA. Oracle can start instances underconfigured and allow the instance to use more memory by growing the SGA components, up to a maximum of `SGA_MAX_SIZE.`
- The memory allocated for an instance's SGA is displayed on instance startup when using Oracle Enterprise Manager (or SQL*Plus). You can also display the current instance's SGA size by using the SQL*Plus `SHOW` statement with the `SGA` clause.

The Database Buffer Cache

- The database buffer cache is the portion of the SGA that holds copies of data blocks read from datafiles.
- All user processes concurrently connected to the instance share access to the database buffer cache.
- Organization of the Database Buffer Cache:

  The buffers in the cache are organized in two lists;

  Write list holds dirty buffers, which contain data that has been modified but has not yet been written to disk.

  and

  Least recently used (LRU) list holds free buffers, pinned buffers, and dirty buffers that have not yet been moved to the write list. *Free buffers* do not contain any useful data and are available for use. *Pinned buffers* are currently being accessed.

Database Buffer Cache Contd…

- The LRU Algorithm and Full Table Scans:
  - When the user process is performing a full table scan, it reads the blocks of the table into buffers and puts them on the LRU end (instead of the MRU end) of the LRU list.
  - You can control this default behavior of blocks involved in table scans on a table-by-table basis. To specify that blocks of the table are to be placed at the MRU end of the list during a full table scan, use the `CACHE` clause when creating or altering a table or cluster.

Database Buffer Cache Contd…

- Oracle9*i*, Release 1 (9.0.1), supports multiple block size in a database. This is the default block size—the block size used for the system tablespace. You specify the standard block size by setting the initialization parameter `DB_BLOCK_SIZE.`
- To specify the size of the standard block size cache, you set the initialization parameter `DB_CACHE_SIZE.`
- You can also set the size for two additional buffer pools, `KEEP` and `RECYCLE`, by setting `DB_KEEP_CACHE_SIZE` and `DB_RECYCLE_CACHE_SIZE.`

Database Buffer Cache Contd…

- Multiple Buffer Pools
  - You can configure the database buffer cache with separate buffer pools that either keep data in the buffer cache or make the buffers available for new data immediately after using the data blocks.
  - The `KEEP` buffer pool retains the schema object's data blocks in memory.
  - The `RECYCLE` buffer pool eliminates data blocks from memory as soon as they are no longer needed.
  - The `DEFAULT` buffer pool contains data blocks from schema objects that are not assigned to any buffer pool, as well as schema objects that are explicitly assigned to the `DEFAULT` pool.

The Redo Log Buffer

- The **redo log buffer** is a circular buffer in the SGA that holds information about changes made to the database.
- This information is stored in **redo entries**. Redo entries contain the information necessary to reconstruct, or redo, changes made to the database by `INSERT, UPDATE, DELETE, CREATE, ALTER,` or `DROP` operations.
- Redo entries are used for database recovery, if necessary.
- Redo entries are copied by Oracle server processes from the user's memory space to the redo log buffer in the SGA.
- The redo entries take up continuous, sequential space in the buffer.

The Shared Pool

- The shared pool portion of the SGA contains three major areas: library cache, dictionary cache, buffers for parallel execution messages, and control structures.
- The total size of the shared pool is determined by the initialization parameter `SHARED_POOL_SIZE.` The default value of this parameter is 8M on 32-bit platforms and 64M on 64-bit platforms.

The Shared Pool Contd…

- **Library Cache:** The library cache includes the shared SQL areas, private SQL areas (in the case of a multiple transaction server), PL/SQL procedures and packages, and control structures such as locks and library cache handles.
  - Shared SQL areas are accessible to all users, so the library cache is contained in the shared pool within the SGA.
  - Oracle recognizes when two users are executing the same SQL statement and reuses the shared SQL area for those users.
- **Dictionary Cache:**The data dictionary is accessed so often by Oracle that two special locations in memory are designated to hold dictionary data. One area is called the *data dictionary cache*, also known as the *row cache* because it holds data as rows instead of buffers (which hold entire blocks of data). The other area in memory to hold dictionary data is the library cache.

The Large Pool

- **Large pool** provide large memory allocations for:
  - Session memory for the shared server and the Oracle XA interface
  - I/O server processes
  - Oracle backup and restore operations
  - Parallel execution message buffers, if the initialization parameter `PARALLEL_AUTOMATIC_TUNING` is set to `TRUE` (otherwise, these buffers are allocated to the shared pool)
- The large pool does not have an LRU list. It is different from reserved space in the shared pool, which uses the same LRU list as other memory allocated from the shared pool.

Program Global Areas (PGA)

- A program global area (PGA) is a memory region which contains data and control information for a server process.
- Access to it is exclusive to that server process and is read and written only by Oracle code acting on behalf of it.
- The total PGA memory allocated by each server process attached to an Oracle instance is also referred to as the *aggregated PGA memory* allocated by the instance.

Content of the PGA

- Private SQL Area
  - A private SQL area contains data such as bind information and runtime memory structures.
  - Each user that submits the same SQL statement has his or her own private SQL area that uses a single shared SQL area.
  - The private SQL area of a cursor is itself divided into two areas whose lifetimes are different:
    - The persistent area, which contains, for example, bind information. It will be freed only when the cursor is closed.
    - The run-time area, which will be freed when the execution is terminated.

Content of the PGA Contd…

- Cursors and SQL Areas
  - The application developer of an Oracle precompiler program program can explicitly open *cursors*, or handles to specific private SQL areas, and use them as a named resource throughout the execution of the program.
  - Recursive cursors that Oracle issues implicitly for some SQL statements also use shared SQL areas.
- Session Memory
  - **Session memory** is the memory allocated to hold a session's variables (logon information) and other information related to the session.
  - For a shared server, the session memory is shared and not private.

SQL Work Areas

- For complex queries (for example, decision-support queries), a big portion of the runtime area is dedicated to work areas allocated by memory-intensive operators.
- If the amount of data to be processed by these two operators does not fit into a work area, the input data is divided into smaller pieces. This allows some data pieces to be processed in memory while the rest are spilled to temporary disk storage to be processed later.
- The size of a work area can be controlled and tuned. Generally, bigger work areas can significantly improve the performance of a particular operator at the cost of higher memory consumption.

PGA Memory Management for Dedicated Mode

- The various `*_area_size` parameters are hard to tune under the best of circumstances.
- Oracle9*i*, Release 1 (9.0.1), introduces a new mode to automatically and globally manage the size of SQL work areas. To enable this mode, the DBA simply needs to specify the total size dedicated to PGA memory for the Oracle instance. This is done be setting the new initialization parameter `pga_aggregate_target.`

PGA Memory Management Contd…

- The specified number (for example, 2G) is a global target for the Oracle instance and Oracle will try to ensure that the total amount of PGA memory allocated across all database server processes never exceeds this target.
- The resulting PGA memory is then allotted to individual active work areas on the basis of their specific memory requirement.

Dedicated and Shared Servers

Memory Area	Dedicated Server	Shared Server
Nature of session memory	Private	Shared
Location of the persistent area	PGA	SGA
Location of part of the runtime area for `SELECT` statements	PGA	SGA
Location of the runtime area for DML/DDL statements	PGA	PGA

Software Code Areas

- *Software code areas* are portions of memory used to store code that is being executed or can be executed.
- Software areas are usually static in size, changing only when software is updated or reinstalled. The required size of these areas varies by operating system.
- Software areas are read-only and can be installed shared or non-shared. When possible, Oracle code is shared so that all Oracle users can access it without having multiple copies in memory. This results in a saving of real main memory and improves overall performance.

Processes - Introduction

- A ***process*** *(job/task)* is a "thread of control" or a mechanism in an operating system that can execute a series of steps.
- All users must execute two modules of code to access an Oracle database instance:
  - Application or Oracle tool *(issues SQL Queries to an Oracle Database e.g.SQL*Plus provided by Oracle.)*
  - Oracle server code *(interprets and processes the application's SQL statements)*.
- These code modules are executed by processes.

Types of Processes

- The processes in an Oracle system can be categorized into two major groups:
  - ***User processes*** execute the application or Oracle tool code.
  - ***Oracle processes*** execute the Oracle server code. *(They include server processes and background processes)*.
- The ***process structure*** varies for different Oracle configurations, depending on the operating system and the choice of Oracle options.

Process Structure

- The code for connected users can be configured in one of the following ways:
- ***Dedicated server*** - For each user, the database application is run by a different process (a *user process*) than the one that executes the Oracle server code (a <u>*dedicated server process*</u>).
- ***Shared server*** - The database application is run by a different process (a *user process*) than the one that executes the Oracle server code. Each server process that executes Oracle server code (a <u>*shared server process*</u>) can serve multiple user processes.

User Processes

- When a user runs an application program *(Pro*C program, SQL*Plus etc)*, Oracle creates a ***user process*** to run the user's application.
- ***Connection -***A ***connection*** is a *communication pathway* between a user process and an Oracle instance.
- ***Session*** - A ***session*** is a specific connection of a user to an Oracle instance through a user process.

Oracle Processes

Server Processes

- Oracle creates ***server processes*** to handle the requests of user processes connected to the instance.
- The ***server process*** can perform one or more of the following:
  - Parse and execute SQL statements issued
  - Read necessary data blocks from datafiles on disk into the shared database buffers of the SGA, if the blocks are not already present in the SGA
  - Return results in such a way that the application can process the information

Oracle Processes cont..

Background Processes

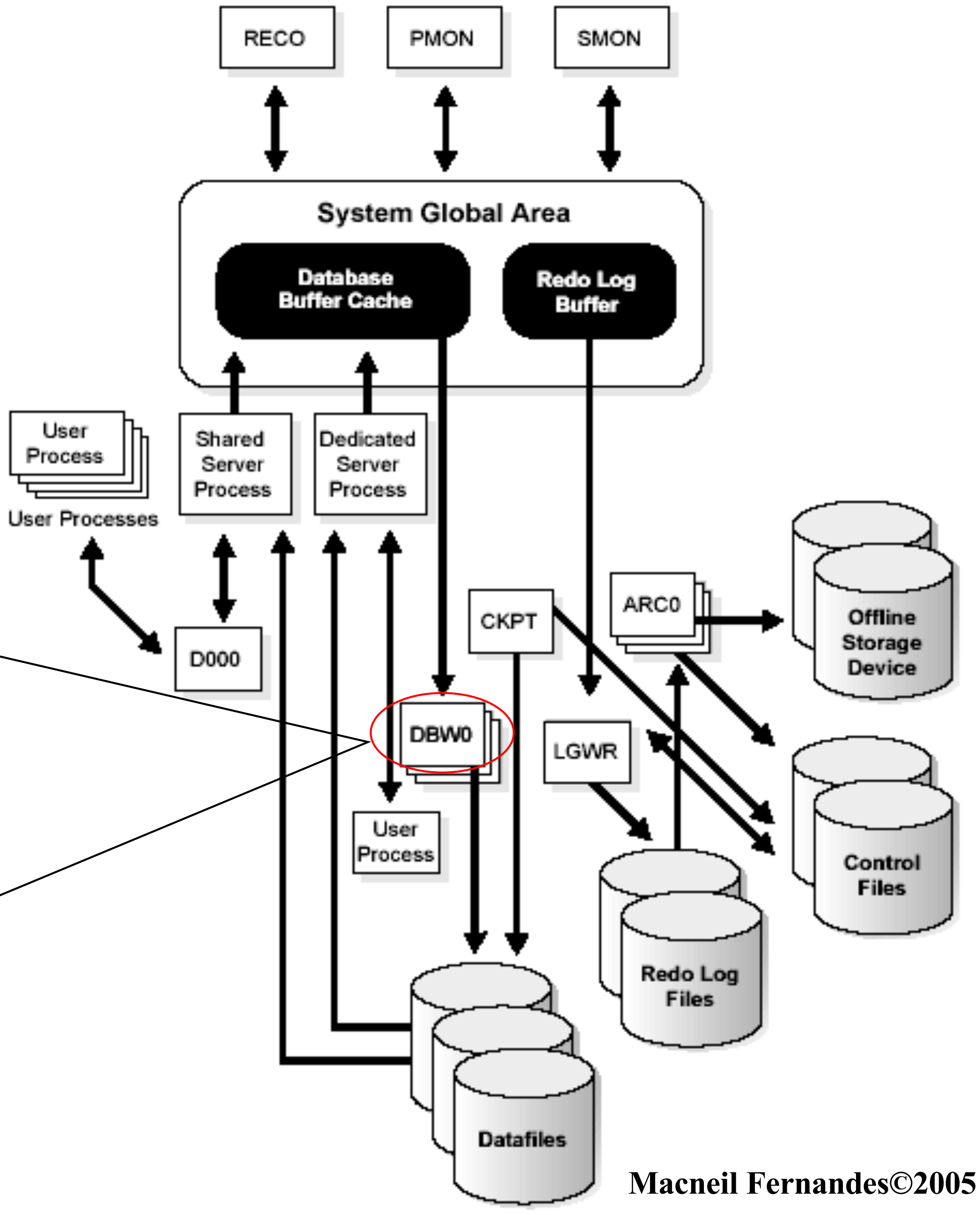
- To maximize performance and accommodate many users, a *multiprocess Oracle system* uses additional Oracle processes called ***background processes***.
- The ***background processes*** include:

  - Database Writer (DBW0/DBW*n*)
  - Log Writer (LGWR)
  - Checkpoint (CKPT)
  - System Monitor (SMON)
  - Process Monitor (PMON)
  - Archiver (ARC*n*)
  - Recoverer (RECO)
  - Lock Manager Server (LMS)
  - Queue Monitor (QMN*n*)
  - Dispatcher (D*nnn*)
  - Server (S*nnn*)

Background Processes

Database Writer Process (DBWn)

- The **database writer process (DBWn)** writes the contents of buffers to datafiles. The DBW*n* processes are responsible for writing modified (dirty) buffers in the database buffer cache to disk.
- When a buffer in the database buffer cache is modified, it is marked **dirty**.
- A **cold** buffer is a buffer that has not been recently used according to the least recently used (LRU) algorithm.
- The DBW*n* process writes cold, dirty buffers to disk so that user processes are able to find cold, clean buffers that can be used to read new blocks into the cache.

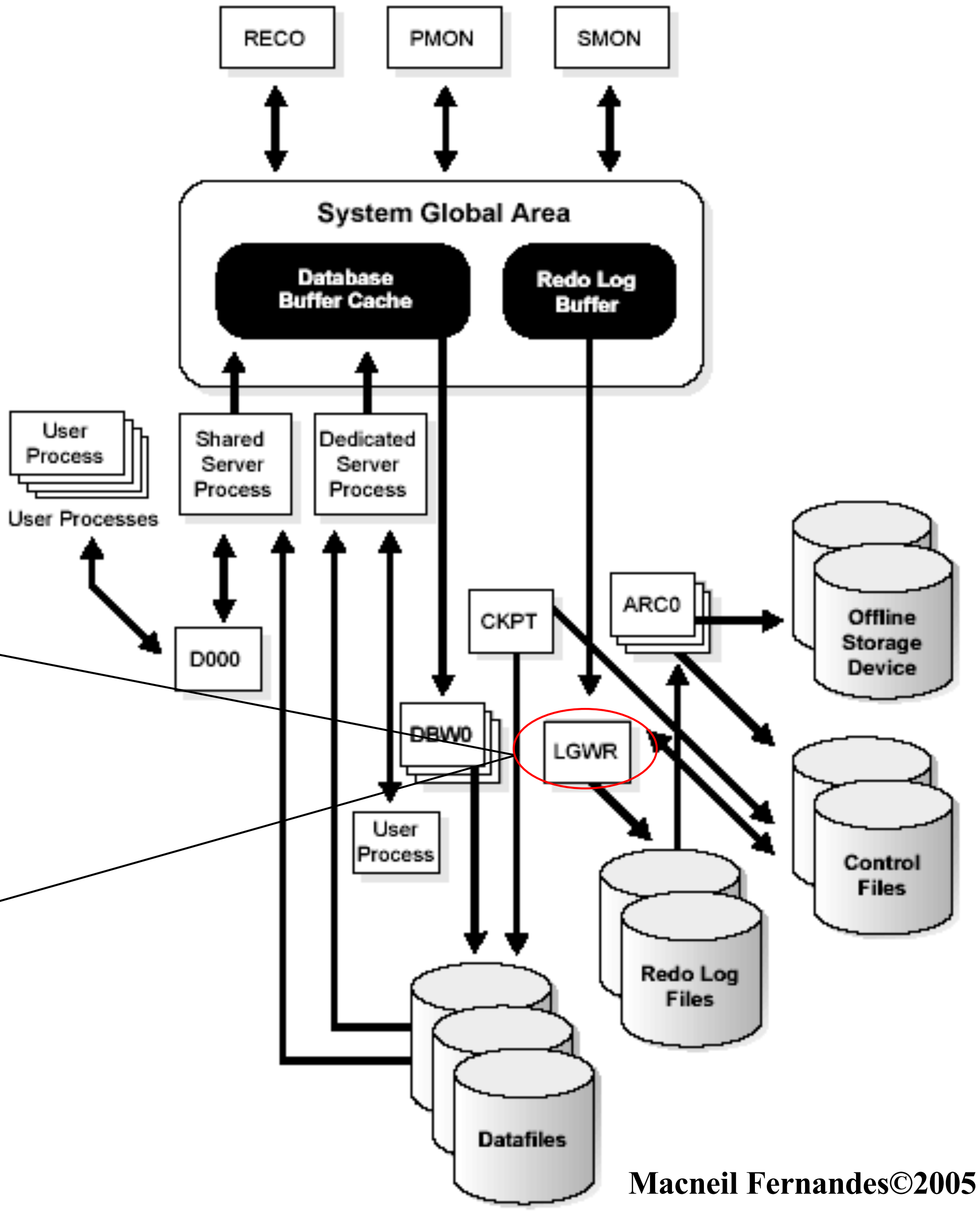


Macneil Fernandes©2005

Background Processes cont..

Log Writer Process (LGWR)

- The **log writer process (LGWR)** is responsible for redo log buffer management i.e.writing the redo log buffer to a redo log file on disk
- **LGWR** writes:
  - A commit record when a user process commits a transaction
  - Redo log buffers
  - – Every 3 seconds
  - – When redo log buffer is 1/3 full
  - – When *DBWn* process writes modified buffers to disk.
- When a user issues a *`COMMIT`* statement, *LGWR* puts a commit record in the redo log buffer and writes it to disk immediately, along with the transaction's redo entries.

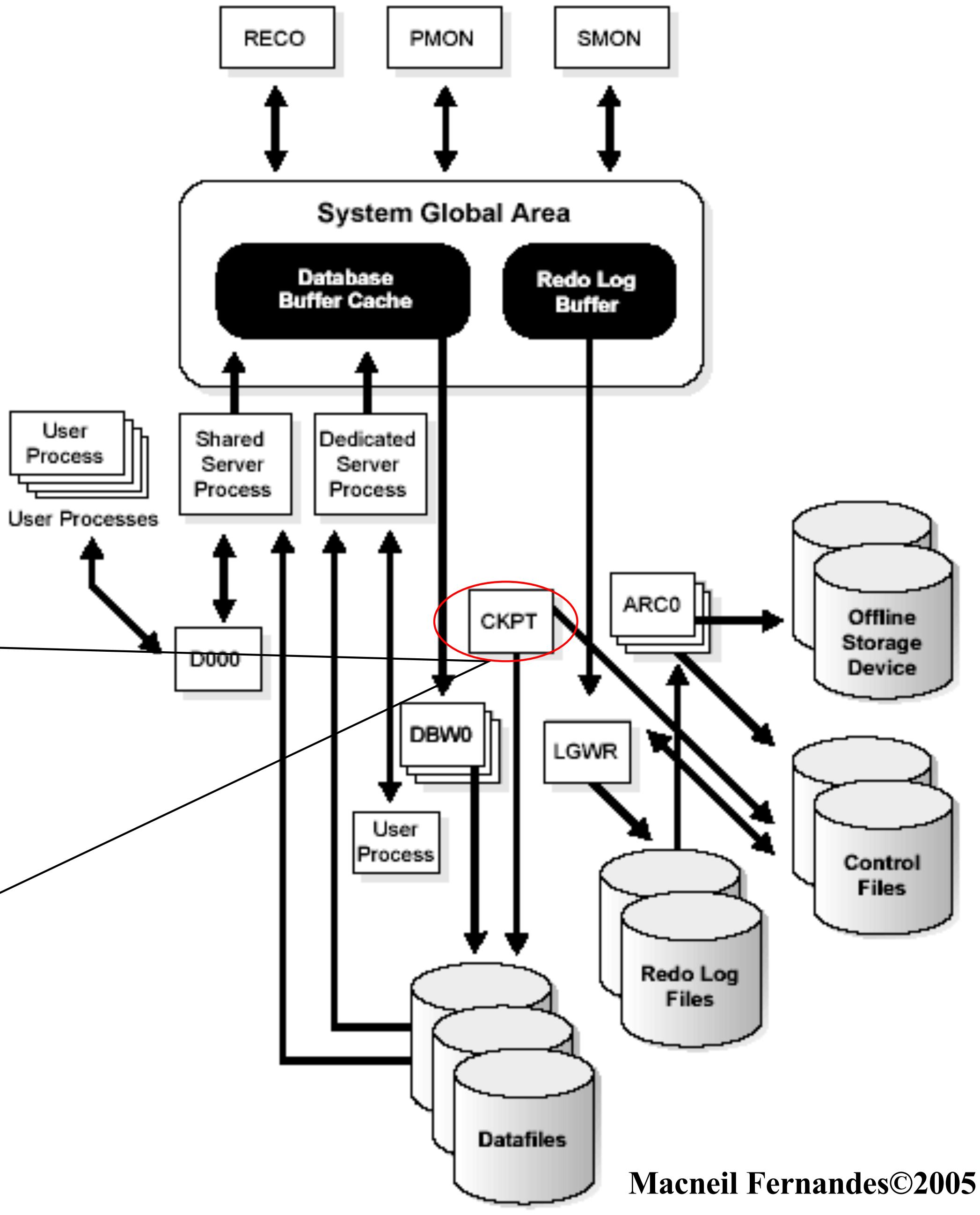


Macneil Fernandes©2005

Background Processes cont..

Checkpoint Process (CKPT)

- When a checkpoint occurs, Oracle must update the headers of all datafiles to record the details of the checkpoint. This is done by the *CKPT* process.

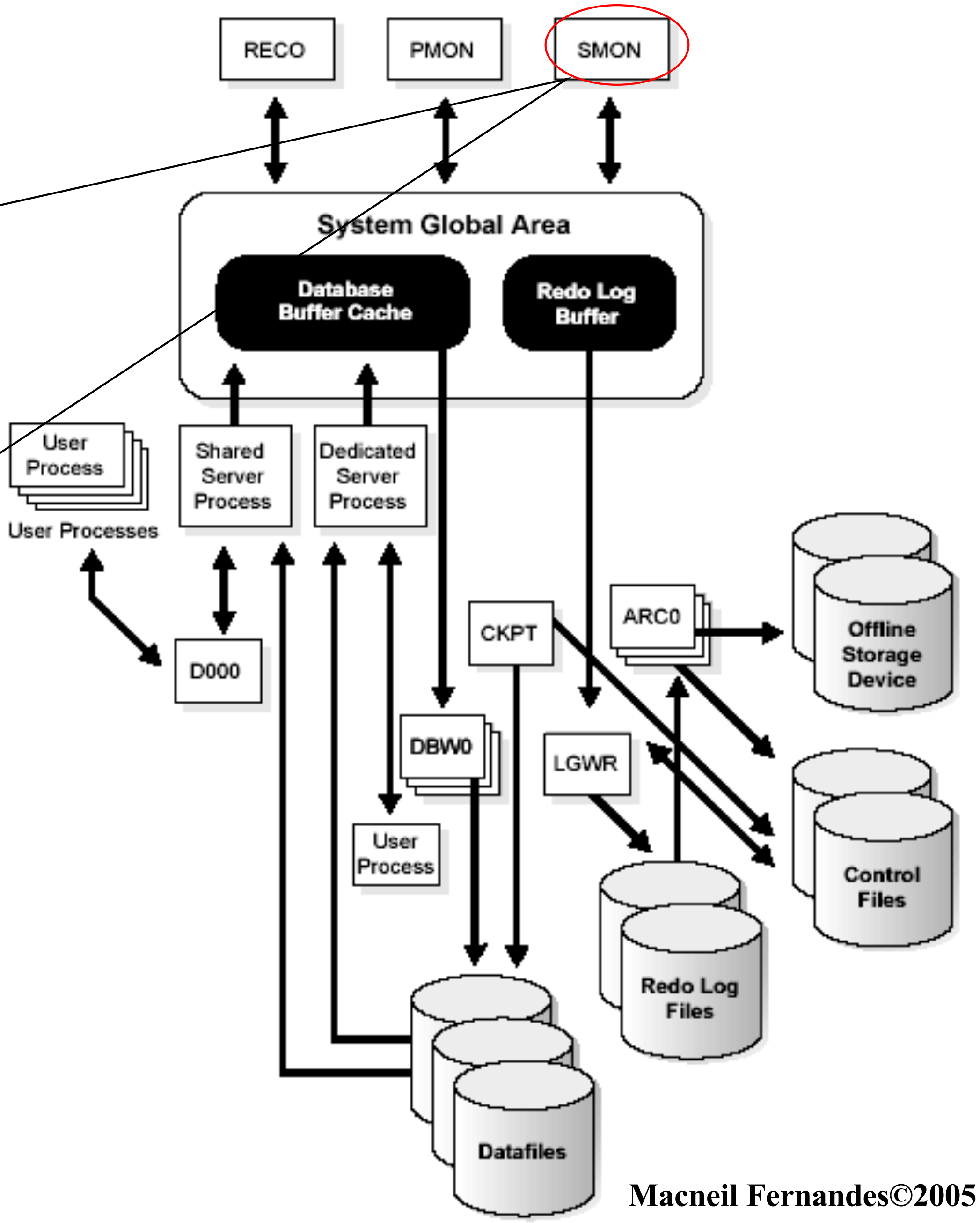


Macneil Fernandes©2005

Background Processes cont..

System Monitor Process (SMON)

- The **system monitor process (SMON)** performs crash recovery, if necessary, at instance startup.
- **SMON** is also responsible for cleaning up temporary segments that are no longer in use and for coalescing contiguous free extents within dictionary-managed tablespaces.
- If any dead transactions were skipped during crash and instance recovery because of file-read or offline errors, **SMON** recovers them when the tablespace or file is brought back online.

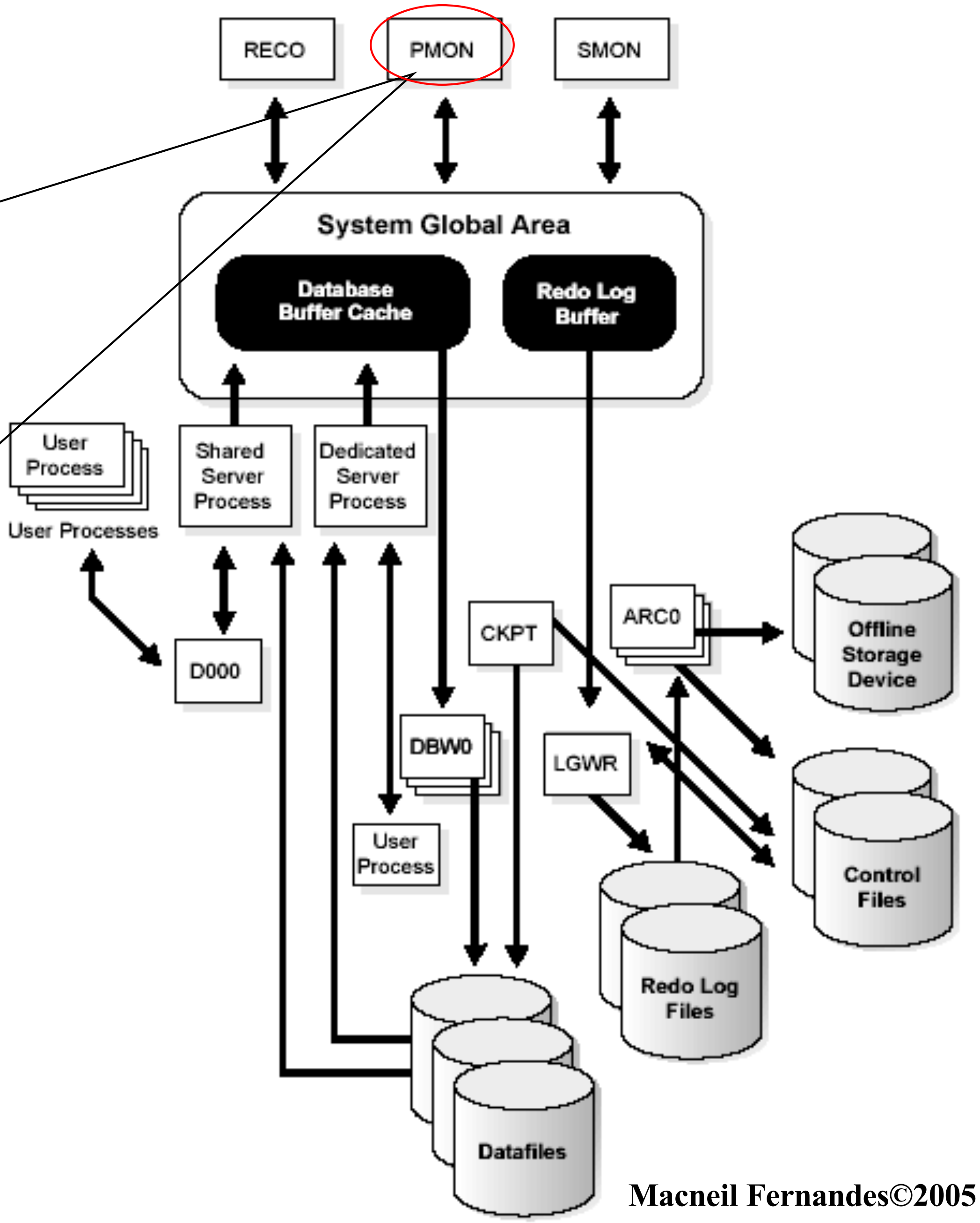


Macneil Fernandes©2005

Background Processes cont..

Process Monitor Process (PMON)

- The **process monitor (PMON)** performs process recovery when a user process fails.
- PMON is responsible for cleaning up the database buffer cache and freeing resources that the user process was using. For example, it resets the status of the active transaction table, releases locks, and removes the process ID from the list of active processes.
- PMON periodically checks the status of dispatcher and server processes, and restarts any that have died

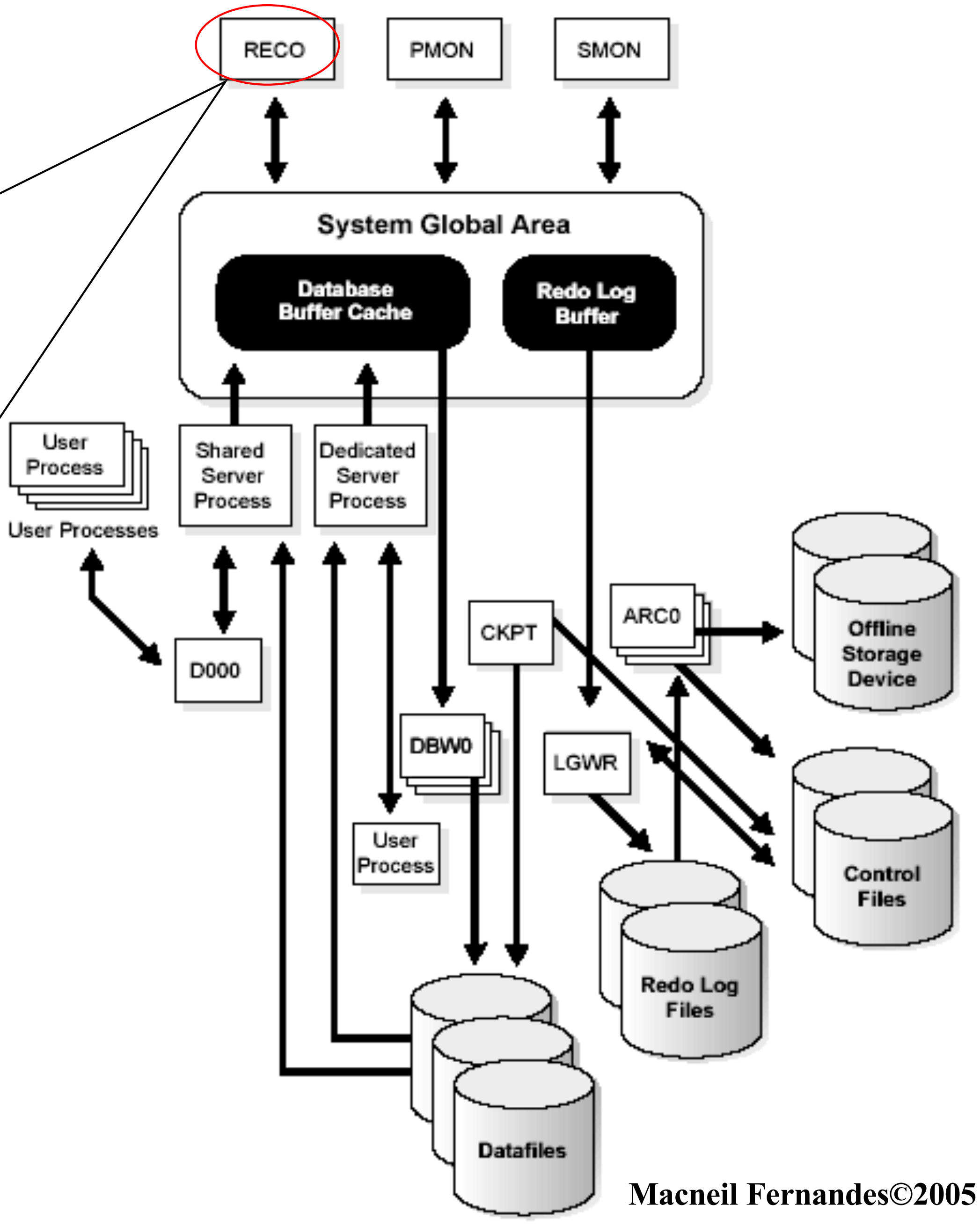


Macneil Fernandes©2005

Background Processes cont..

Recoverer Process (RECO)

- The **recoverer process (RECO)** is a background process used with the distributed database configuration tha automatically resolves failures involving distributed transactions.
- The RECO process of a node automatically connects to other databases involved in an in-doubt distributed transaction. When the RECO process reestablishes a connection between involved databas servers, it automatically resolves all in-doubt transactions, removing from each database's pending transaction table any rows that correspond to th resolved in-doubt transactions.
- If the RECO process fails to connec with a remote server, RECO automatically tries to connect again after a timed interval.

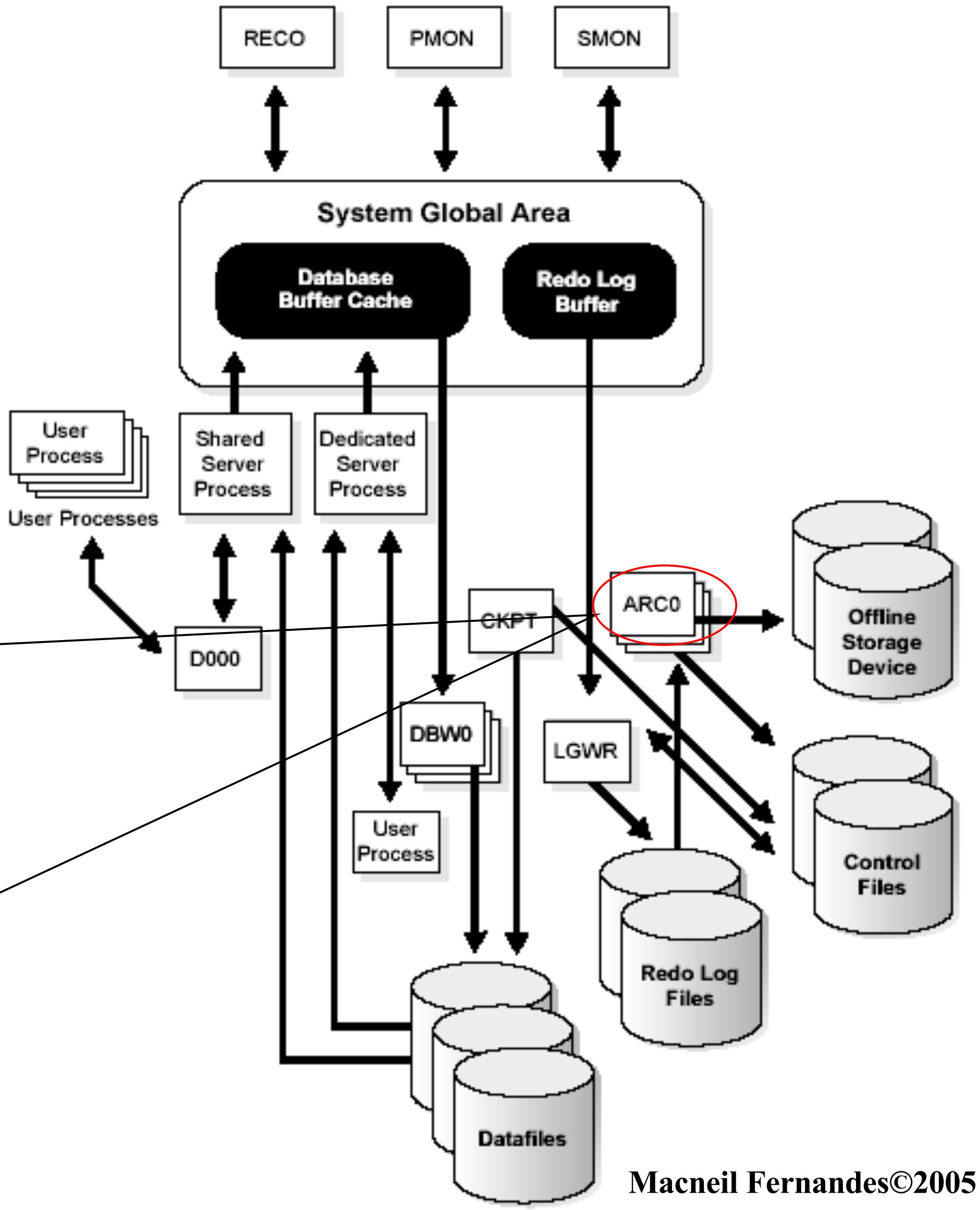


Macneil Fernandes©2005

Background Processes cont..

Archiver Processes (ARC*n*)

- The **archiver process (ARCn)** copies online redo log files to a designated storage device after a log switch has occurred.
- ARC*n* processes are present only when the database is in ARCHIVELOG mode, and automatic archiving is enabled.



Other Background Processes

Lock Manager Server Process (LMS)

- In Oracle9*i* Real Application Clusters, a Lock Manager Server process (LMS) provides inter-instance resource management.

Queue Monitor Processes (QMN*n*)

- The **queue monitor process** is an optional background process for Oracle Advanced Queuing, which monitors the message queues.

Trace Files and the Alert Log

- ***Trace File*** - Each server and background process can write to an associated *trace file*. When a process detects an internal error, it dumps information about the error to its *trace file*.
- ***Alert log*** - The ALERT file of a database is a chronological log of messages and errors, including the following:
  - All internal errors, block corruption errors, and deadlock errors that occur
  - Administrative operations, such as the SQL statements `CREATE/ALTER/DROP DATABASE/ TABLESPACE/ ROLLBACK SEGMENT` and the Oracle Enterprise Manager or SQL*Plus statements `STARTUP, SHUTDOWN, ARCHIVE LOG,` and `RECOVER`
  - Several messages and errors relating to the functions of shared server and dispatcher processes
  - Errors during the automatic refresh of a materialized view

Shared Server Architecture

- ***Shared server*** architecture eliminates the need for a dedicated server process for each connection.
- A *dispatcher* directs multiple incoming network session requests to a *pool of shared server processes*.
- An idle *shared server process* from a shared pool of server processes picks up a request from a common queue, which means a small number of shared servers can perform the same amount of processing as many dedicated servers.
- Also, because the amount of memory required for each user is relatively small, less memory and process management are required, and more users can be supported.

Shared Server Architecture cont..

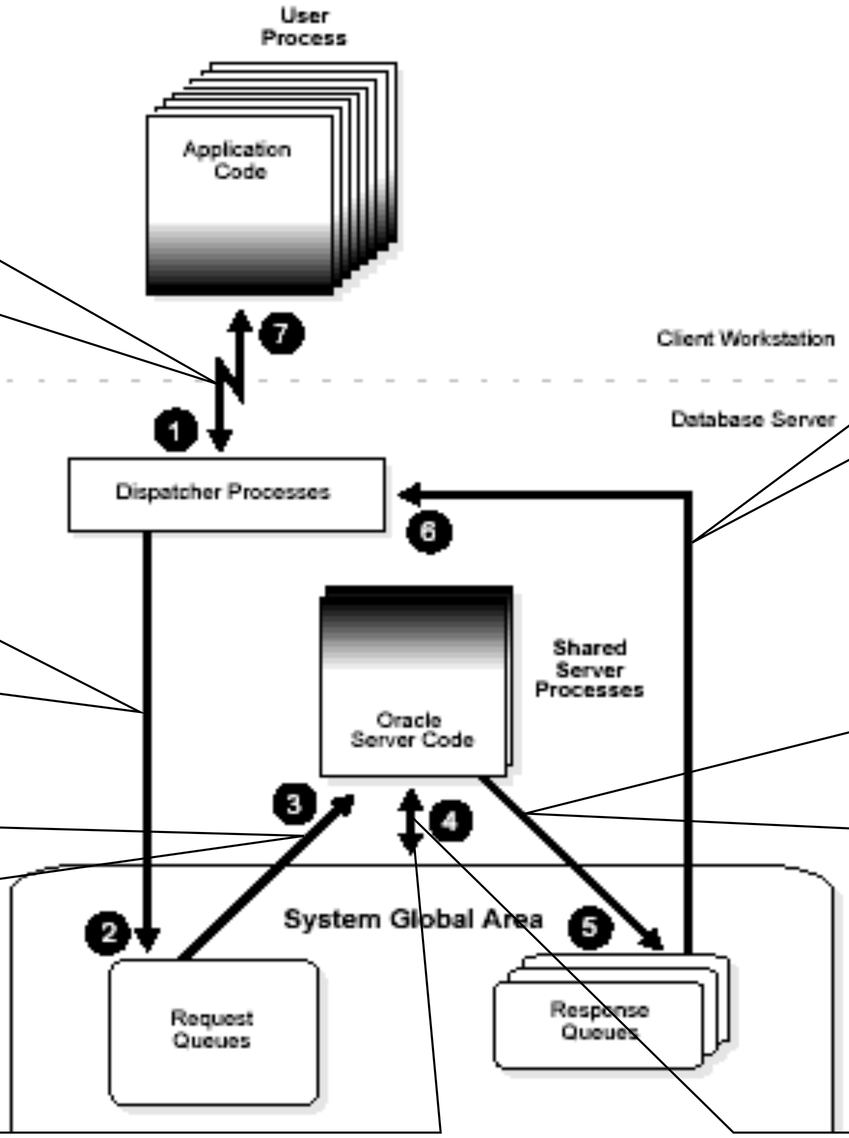
- A number of different processes are needed in a shared server system:
    - A *network listener* process that connects the user processes to dispatchers or dedicated servers.
    - One or more *dispatcher processes*
    - One or more *shared server processes*
- When an instance starts, the *network listener process* establishes a communication pathway.
- Each *dispatcher process* gives the *listener process* an address at which the dispatcher listens for connection requests.
- When a *user process* makes a connection request, the *listener* examines the request and determines whether the user process can use a *shared server process*. If so, the listener returns the address of the dispatcher process that has the lightest load, and the user process connects to the dispatcher directly.

Shared Server Architecture cont..

- A request from a user is a single program interface call that is part of the user's SQL statement.

- The Dispatcher Process places the request on the request queue, where it is picked up by the next available shared server process.

- The request from the Request queue is picked up by the next available shared server process.



- The dispatcher then returns the completed request to the appropriate user process.

- When the server completes the request, it places the response on the calling dispatcher's **response queue**. Each dispatcher has its own response queue in the SGA.

- The shared server processes check the common request queue for new requests, picking up new requests on a first-in-first-out basis. One shared server process picks up one request in the queue and makes all necessary calls to the database to complete that request.

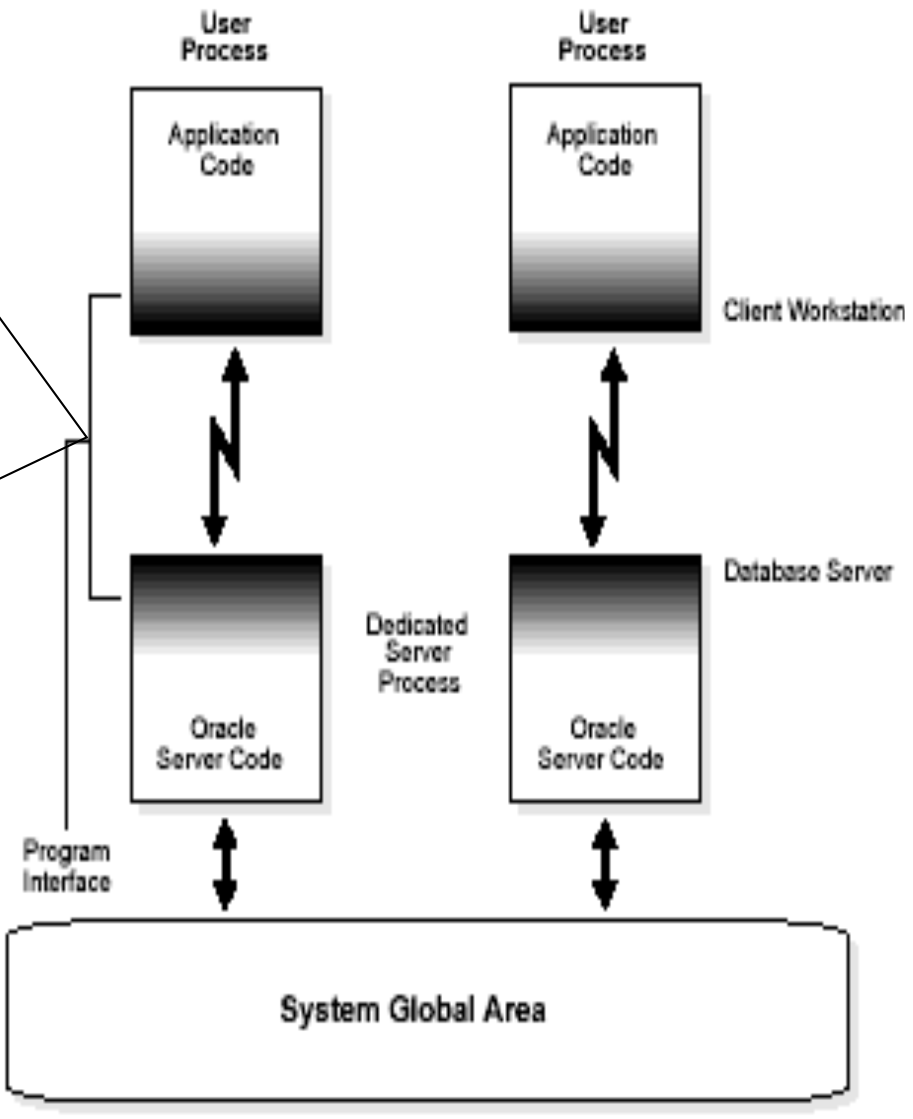
Dedicated Server Configuration

- In this configuration, a user process executes the database application on one machine, and a server process executes the associated Oracle server on another machine.
- The user and server processes are separate, distinct processes. The separate server process created on behalf of each user process is called a ***dedicated server process*** because this server process acts only on behalf of the associated user process.
- This configuration maintains a one-to-one ratio between the number of user processes and server processes. Even when the user is not actively making a database request, the dedicated server process remains (though it is inactive and can be paged out on some operating systems).

Dedicated Server Configuration

The Program Interface

- The **program interface** is the software layer between a database application and Oracle. The program interface:
  - Provides a security barrier
  - Acts as a communication mechanism, formatting requests, passing data, and trapping and returning errors
- The program interface consists of :
- Oracle call interface (OCI) or the Oracle runtime library (SQLLIB)
- The client or user side of the program interface (also called the **UPI**)
- Various **Oracle Net Services drivers**
- Operating system communications software
- The server or Oracle side of the program interface (also called the *OPI*)



Macneil Fernandes©2005

Schemas and Schema Objects

- A **schema** is a collection of database objects that are available to a user.
- **Schema objects** are the logical structures that directly refer to the database’s data. Schema objects include such structures as tables, views, sequences, stored procedures,synonyms, indexes, clusters, and database links.

Tables

Tables are the basic unit of data storage in an Oracle database. Data is stored in **rows** and **columns**. A table is defined with a **table name.** Each column has a **column name,** a **datatype** and a **width**. If columns are of the NUMBER datatype, define **precision** and **scale** instead of width.

A row is a collection of column information corresponding to a single record.

Rules can be specified for each column of a table. These rules are called **integrity constraints**. One example is a `NOT NULL` integrity constraint. This constraint forces the column to contain a value in every row.

How Table Data Is Stored

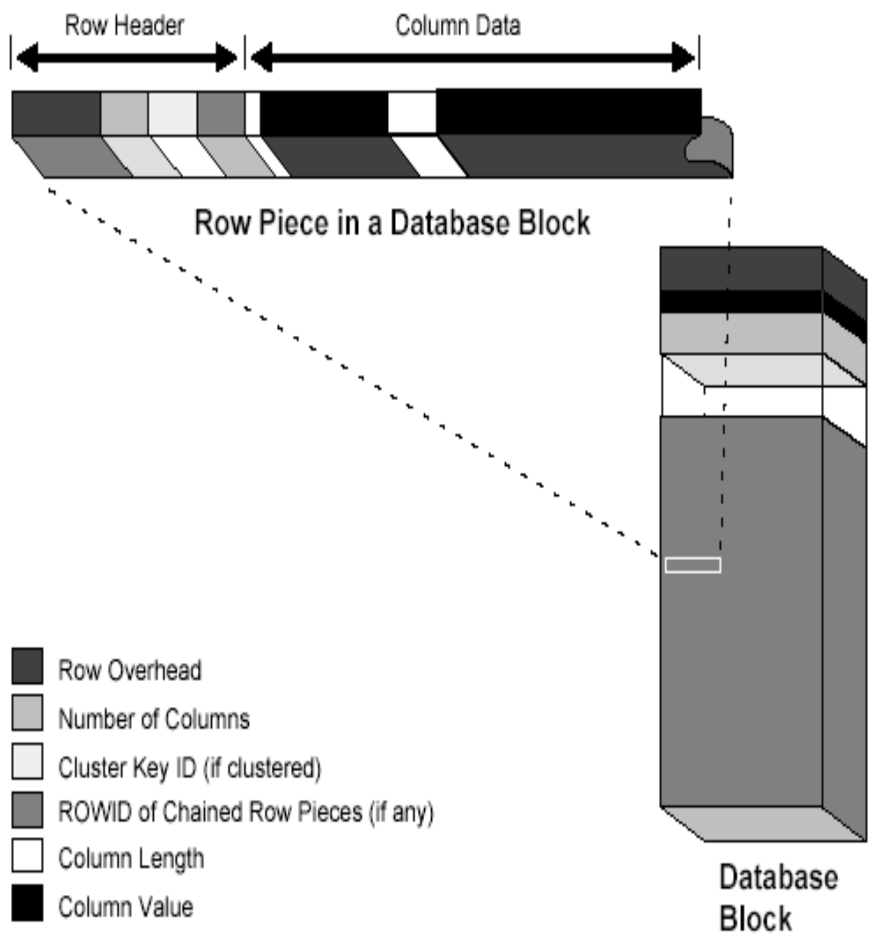
- When you create a table, Oracle automatically allocates a data segment in a tablespace to hold the table’s future data. The allocation of space for a table’s data segment can be controlled and use of this reserved space in the following ways:

- 1.by setting the storage parameters for the data segment.
- 2. by setting the `PCTFREE` and `PCTUSED` parameters for the data segment.

Row Format and Size

- Oracle stores each row of a database table containing data for less than 256 columns as one or more row pieces. If an entire row can be inserted into a single data block,then Oracle stores the row as one row piece. Oracle stores the row using multiple row pieces.

Figure 11–3 The Format of a Row Piece


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

Rowids of Row Pieces

- The **rowid** identifies each row piece by its location or address. Once assigned, a given row piece retains its rowid until the corresponding row is deleted or exported and imported using the Export and Import utilities.

Column Order

- The column order is the same for all rows in a given table. Columns are usually stored in the order in which they were listed in the `CREATE TABLE` statement, but this is not guaranteed. For example, if you create a table with a column of datatype `LONG,` then Oracle always stores this column last. Also, if a table is altered so that a new column is added, then the new column becomes the last column stored.

Partitioned Tables

- Partitioned tables allow your data to be broken down into smaller, more manageable pieces called partitions, or even subpartitions. Indexes can be partitioned in similar fashion. Each partition can be managed individually, and can operate independently of the other partitions, thus providing a structure that can be better tuned for availability and performance.

Nested Tables

- You can create a table with a column whose datatype is another table. That is, tables can be **nested** within other tables as values in a column. The Oracle server stores nested table data out of line from the rows of the parent table, using a **store table** that is associated with the nested table column. The parent row contains a unique set identifier value associated with a nested table instance.

Temporary Tables

- In addition to permanent tables, Oracle can create **temporary tables** to hold session-private data that exists only for the duration of a transaction or session.
- The `CREATE GLOBAL TEMPORARY TABLE` statement creates a temporary table that can be transaction-specific or session-specific. For transaction-specific temporary tables, data exists for the duration of the transaction. For session-specific temporary tables, data exists for the duration of the session. Data in a temporary table is private to the session. Each session can only see and modify its own data

External Tables

- An external table describes how the external table layer needs to present the data to the server. It is the responsibility of the access driver and the external table layer to do the necessary transformations required on the data in the data file so that it matches the external table definition.

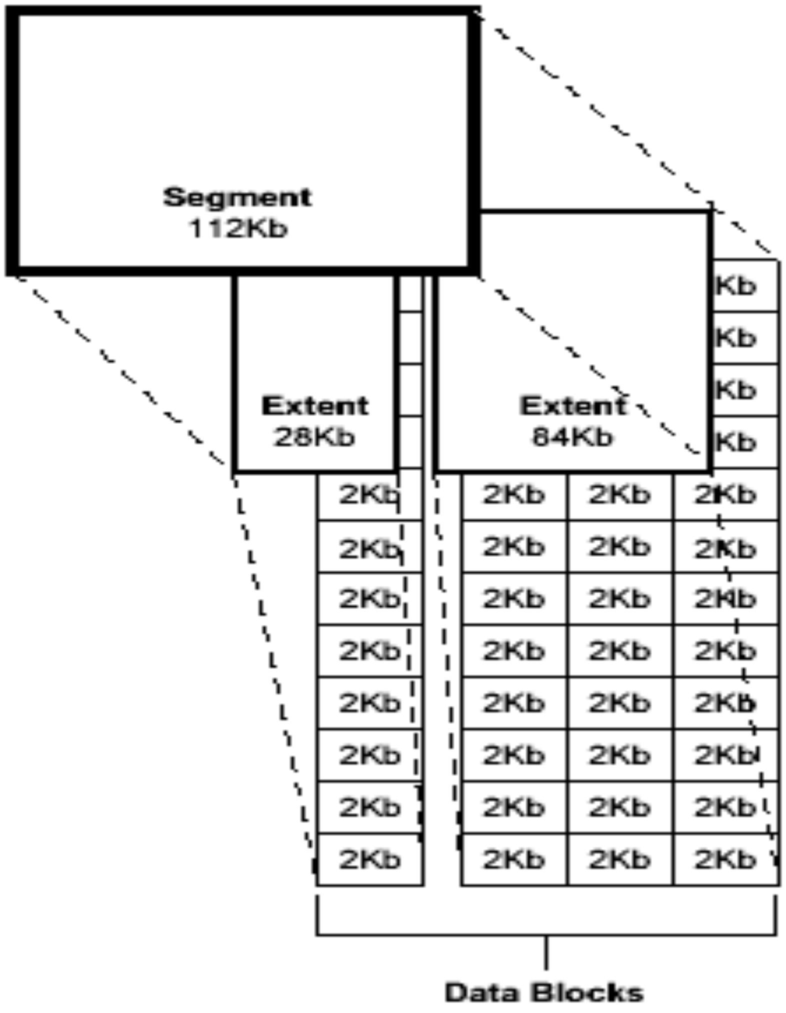
Data Dictionary

- An Oracle data dictionary is a set of tables and views that are used as a **read-only** reference about the database.
- A data dictionary stores information about both the logical and physical structure of the database along with information as:
  - The valid users of a database
  - Information about integrity constraints
  - Amount of space allocated for a schema object and how much of it is in use
- A data dictionary is created whenever a database is created.

Data Blocks, Extents, and Segments

- Oracle stores data in **data blocks.** One data block corresponds to a specific number of bytes of physical database space on disk.
- An **extent** is a specific number of contiguous data blocks allocated for storing a specific type of information.
- A **segment** is a set of extents, each of which has been allocated for a specific data structure and all of which are stored in the same tablespace.
- Oracle allocates space for segments in units of one extent.

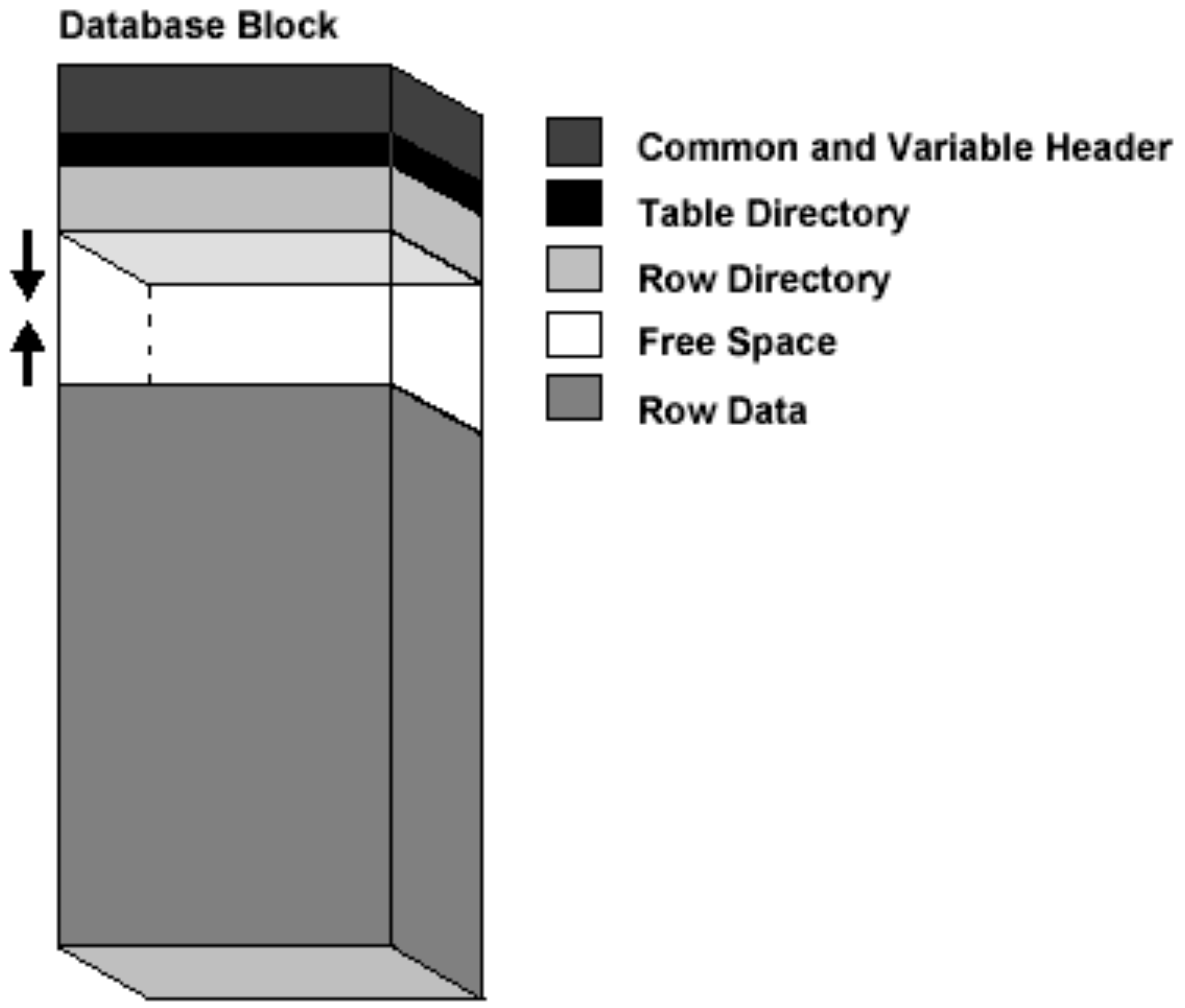
Figure 3–1 The Relationships Among Segments, Extents, and Data Blocks



Data Blocks

- A data block is the smallest unit of I/O used by a database.
- The standard block size is specified by the initialization parameter `DB_BLOCK_SIZE`.
- Oracle requests data in multiples of Oracle data blocks, not operating system blocks (at the Operating System Level all data is stored in bytes).

Data Block Format



Data Block Format

Header (Common and Variable)

The header contains general block information, such as the block address and the type of segment (for example, data, index, or rollback).

Table Directory

This portion of the data block contains information about the table having rows in this block.

Row Directory

This portion of the data block contains information about the actual rows in the block (including addresses for each row piece in the row data area).

Data Block Format

Row Data

This portion of the data block contains table or index data.

Free Space

Free space is allocated for insertion of new rows and for updates to rows that require additional space (for example, when a trailing null is updated to a nonnull value).

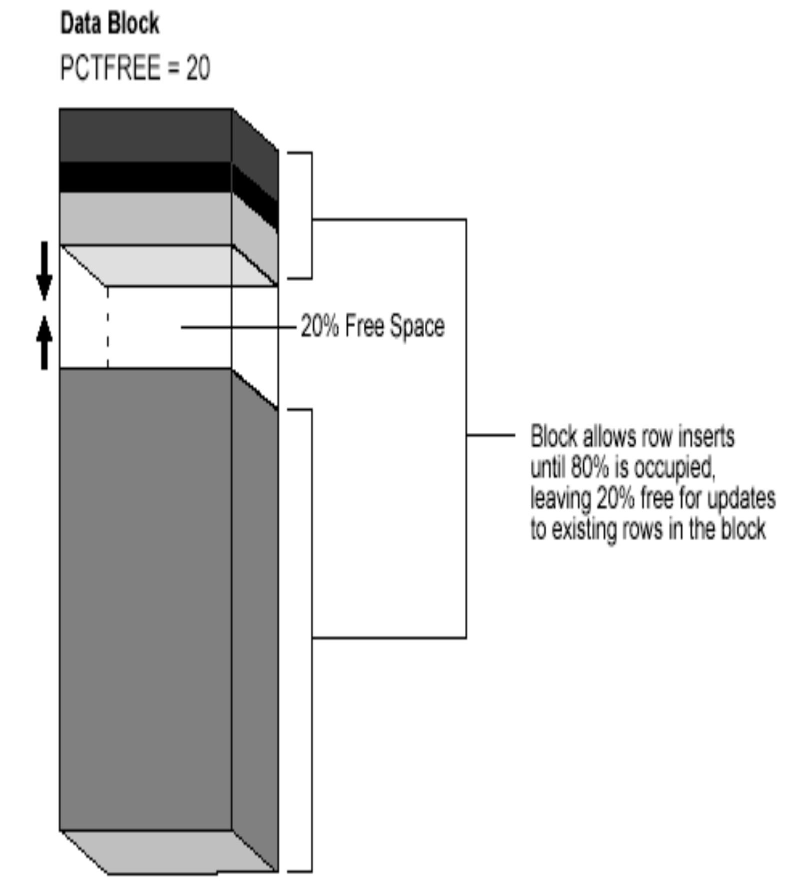
PCTFREE, PCTUSED, and Row Chaining

- For manually managed tablespaces, two space management parameters, `PCTFREE` and `PCTUSED,` enable one to control the use of free space for inserts of and updates to the rows in all the data blocks of a particular segment.

PCTFREE

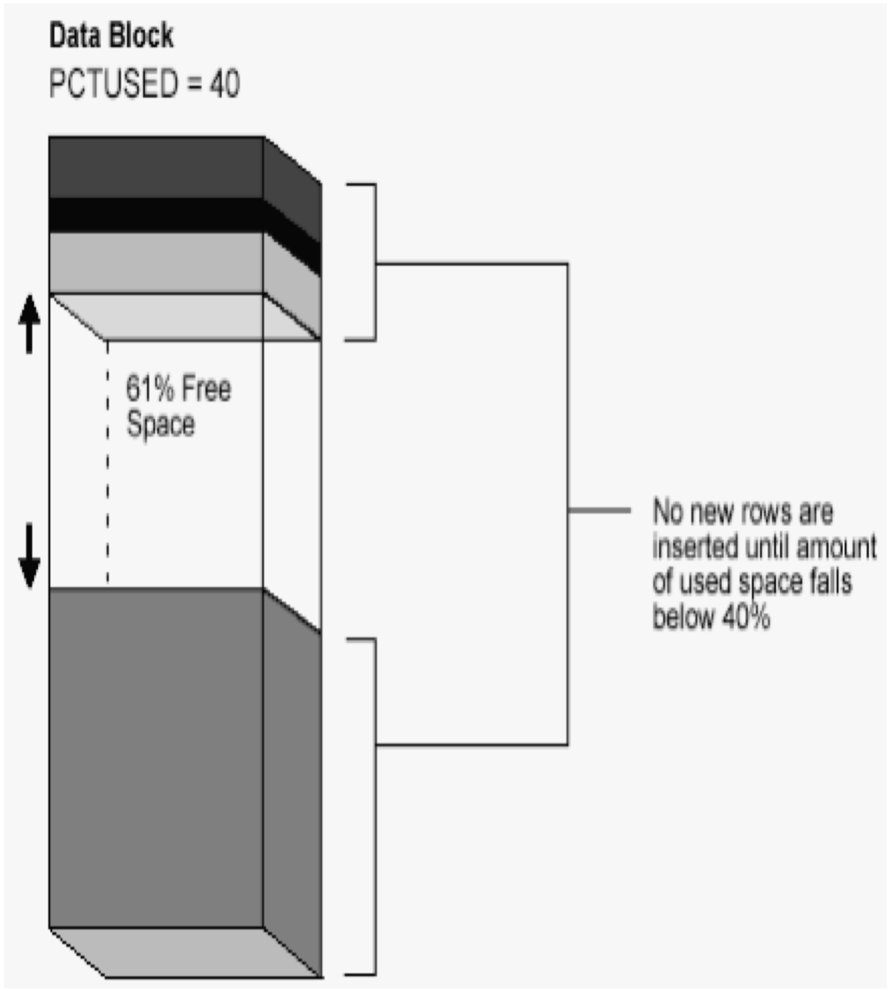
- The `PCTFREE` parameter sets the minimum percentage of a data block to be
- **reserved** as free space for possible updates to rows that already exist in that block.

Figure 3–3 PCTFREE


Data 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

- The `PCTUSED` parameter sets the minimum percentage of a block that can be used for row data plus overhead before new rows are added to the block.

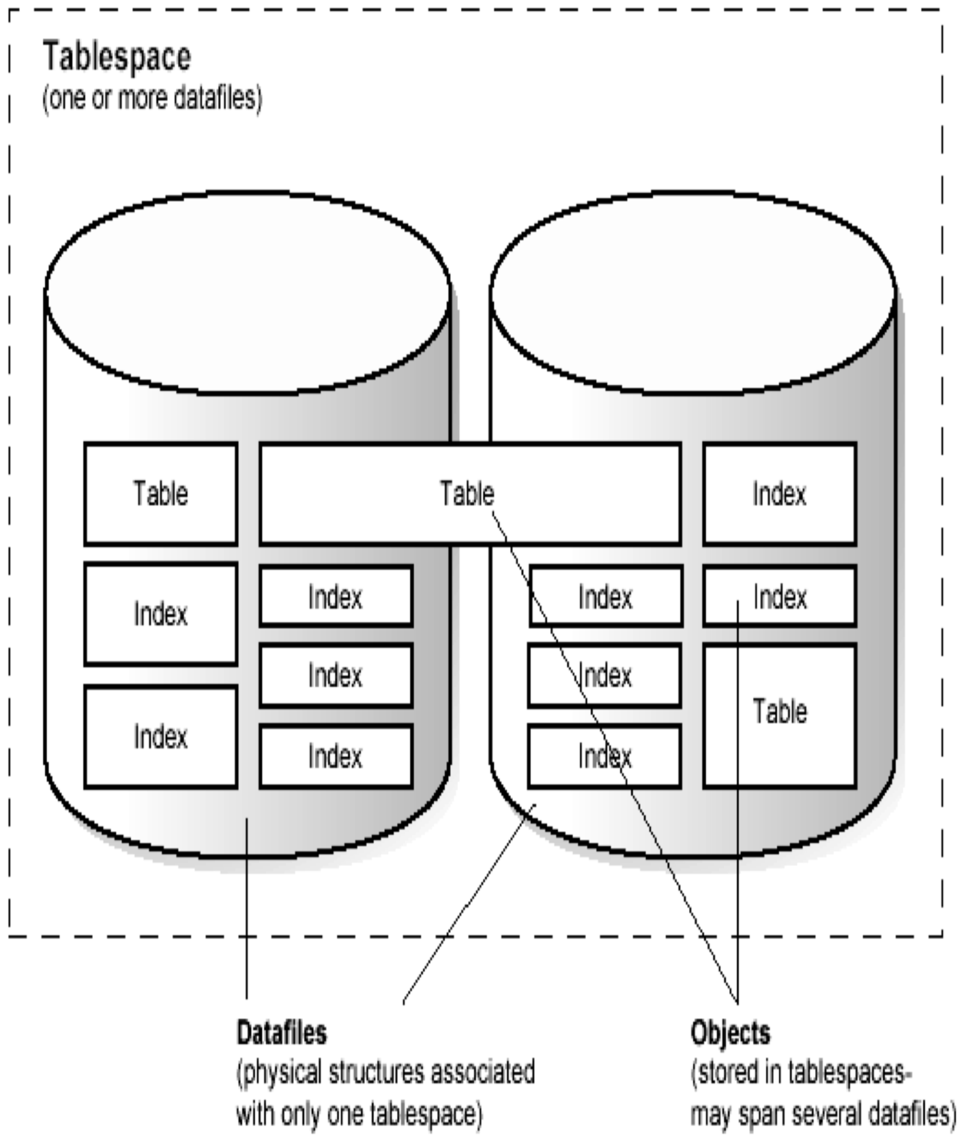

Data Block
PCTUSED = 40
61% Free Space
No new rows are inserted until amount of used space falls below 40%

Segments

- Oracle databases use four types of segments,
  - Data Segments
  - Index Segments
  - Temporary Segments
  - Rollback Segments

Tablespaces, Datafiles, and Control Files

- Oracle stores data logically in **tablespaces** and physically in **datafiles** associated with the corresponding tablespace.



Tablespaces

- A database is divided into one or more logical storage units called tablespaces.
- Tablespaces are divided into logical units of storage called **segments**, which are further divided into **extents**.
- Extents are a collection of contiguous blocks.

SYSTEM Tablespace

- Every Oracle database contains a tablespace named `SYSTEM,` which Oracle creates automatically when the database is created.
- This Tablespace comprises of:
    1. The Data Dictionary
    2. PL/SQL Program Units Description

TableSpaces Cont..

Undo Tablespaces

- *Undo tablespaces* are special tablespaces used solely for storing undo information.

Default Temporary Tablespace

- Oracle stores temporary data for user in the *Default Temporary Tablespace.*

Datafiles

- A tablespace in an Oracle database consists of one or more physical **datafiles**.
- Oracle creates a datafile for a tablespace by allocating the specified amount of disk space plus the overhead required for the file header.

Control Files

- The control file of a database is a small binary file necessary for the database to start and operate successfully.
- A control file is updated continuously by Oracle during database use.

A control file contains:

- The database name
- The timestamp of database creation
- Tablespace information
- Datafile offline ranges
- The log history
- Backup datafile and redo log information

Views

- A view is a tailored presentation of the data contained in one or more tables or other views. A view takes the output of a query and treats it as a table. Therefore, a view can be thought of as a stored query or a virtual table.
- All operations performed on a view actually affect data in some base table of the view and are subject to the integrity constraints and triggers of the base tables.

Use of Views

- Provide an additional level of table security
- Hide data complexity.
- Present the data in a different perspective from that of the base table.
- Store complex queries.

Storage for Views

- Unlike a table, a view is not allocated any storage space, nor does a view actually contain data. Rather, a view is defined by a query that extracts or derives data from the tables that the view references. These tables are called **base tables**. Base tables can in turn be actual tables or can be views themselves. Because a view is based on other objects, a view requires no storage other than storage for the definition of the view (the stored query) in the data dictionary.

Materialized Views

- **Materialized views** are schema objects that can be used to summarize, precompute,replicate, and distribute data. They are suitable in various computing environments such as data warehousing, decision support, and distributed or mobile computing:
- In data warehouses, materialized views are used to precompute and store aggregated data such as sums and averages.
- In distributed environments, materialized views are used to replicate data at distributed sites and synchronize updates done at several sites with conflict resolution methods.

Dimensions

- A dimension is a schema object that defines hierarchical relationships between pairs of columns or column sets.
- A dimension is a container of logical relationships between columns and does not have any data storage assigned to it.

The `CREATE DIMENSION` statement specifies:

- Multiple `LEVEL` clauses, each of which identifies a column or column set in the dimension
- One or more `HIERARCHY` clauses that specify the parent/child relationships between adjacent levels
- Optional `ATTRIBUTE` clauses, each of which identifies an additional column or column set associated with an individual level

Sequence Generator

- The sequence generator provides a sequential series of numbers. The sequence generator is especially useful in multiuser environments for generating unique sequential numbers without the overhead of disk I/O or transaction locking.

  SYNTAX:

  ```
  CREATE SEQUENCE new_seq
  START WITH 100
  INCREMENT BY 1
  CACHE 20
  ORDER;
  ```

Sequence Generator(cont..)

- Sequence numbers are Oracle integers defined in the database of up to 38 digits. A sequence definition indicates general information:
- The name of the sequence
- Whether the sequence ascends or descends
- The interval between numbers
- Whether Oracle should cache sets of generated sequence numbers in memory.

Oracle stores the definitions of all sequences for a particular database as rows in a single data dictionary table in the `SYSTEM` tablespace. Therefore, all sequence definitions are always available, because the `SYSTEM` tablespace is always online

Synonyms

A **synonym** is an alias for any table, view, materialized view, sequence, procedure,function, or package. Because a synonym is simply an alias, it requires no storage other than its definition in the data dictionary.

Synonyms are often used for security and convenience. For example, they can do the following:

- Mask the name and owner of an object
- Provide location transparency for remote objects of a distributed database
- Simplify SQL statements for database users

Sequence Generator(cont..)

You can create both public and private synonyms. A **public** synonym is owned by the special user group named `PUBLIC` and every user in a database can access it. A **private** synonym is in the schema of a specific user who has control over its availability to others.

Synonyms are very useful in both distributed and nondistributed database environments because they hide the identity of the underlying object, including its location in a distributed system. This is advantageous because if the underlying object must be renamed or moved, then only the synonym needs to be redefined.

Indexes

- Indexes are optional structures associated with tables and clusters. You can create indexes on one or more columns of a table to speed SQL statement execution on that table. Just as the index in this manual helps you locate information faster than if there were no index, an Oracle index provides a faster access path to table data.Indexes are the primary means of reducing disk I/O when properly used.

Indexes(cont..)

- You can create many indexes for a table as long as the combination of columns differs for each index. You can create more than one index using the same columns if you specify distinctly different combinations of the columns. For example, the following statements specify valid combinations:
- `CREATE INDEX emp_idx1 ON emp (ename, job);`
- `CREATE INDEX emp_idx2 ON emp (job, ename);`

Unique and Nonunique Indexes

- **Indexes** can be unique or nonunique. Unique indexes guarantee that no two rows of a table have duplicate values in the key column (or columns). Nonunique indexes do not impose this restriction on the column values.
- Alternatively, you can define UNIQUE integrity constraints on the desired columns.Oracle enforces UNIQUE integrity constraints by automatically defining a unique index on the unique key.

Composite Indexes

- A **composite index** is an index that you create on multiple columns in a table. Columns in a composite index can appear in any order and need not be adjacent in the table.
- Composite indexes can speed retrieval of data for `SELECT` statements in which the `WHERE` clause references all or the leading portion of the columns in the composite index. Therefore, the order of the columns used in the definition is important.Generally, the most commonly accessed or most selective columns go first.

Indexes and Keys

- Although the terms are often used interchangeably, there is a distinction between **indexes** and **keys**. **Indexes** are structures actually stored in the database, which users create, alter, and drop using SQL statements. You create an index to provide a fast access path to table data. **Keys** are strictly a logical concept. Keys correspond to another feature of Oracle called integrity constraints, which enforce the business rules of a database.

Indexes and Nulls

- `NULL` values in indexes are considered to be distinct except when all the non-`NULL` values in two or more rows of an index are identical, in which case the rows are considered to be identical. Therefore, `UNIQUE` indexes prevent rows containing `NULL` values from being treated as identical. This does not apply if there are no non-`NULL` values—in other words, if the rows are entirely `NULL.`

Function Based Indexes

- A **function-based index** precomputes the value of the function or expression and stores it in the index.
- Function-based index can be created either as a B-tree or a bitmap index.
- The function used for building the index can be an arithmetic expression or an expression that contains a PL/SQL function, package function, or SQL function.

Uses of Function-Based Indexes

- Function-based indexes provide an efficient mechanism for evaluating statements that contain functions in their WHERE clauses.
- The value of the expression is computed and stored in the index.
- Function-based indexes defined on UPPER(*column_name*) or LOWER(*column name*) can facilitate case-insensitive searches.

  For example, the following index:

  Create INDEX uppercase_idx ON emp (UPPER(empname));

  can facilitate processing queries such as this:

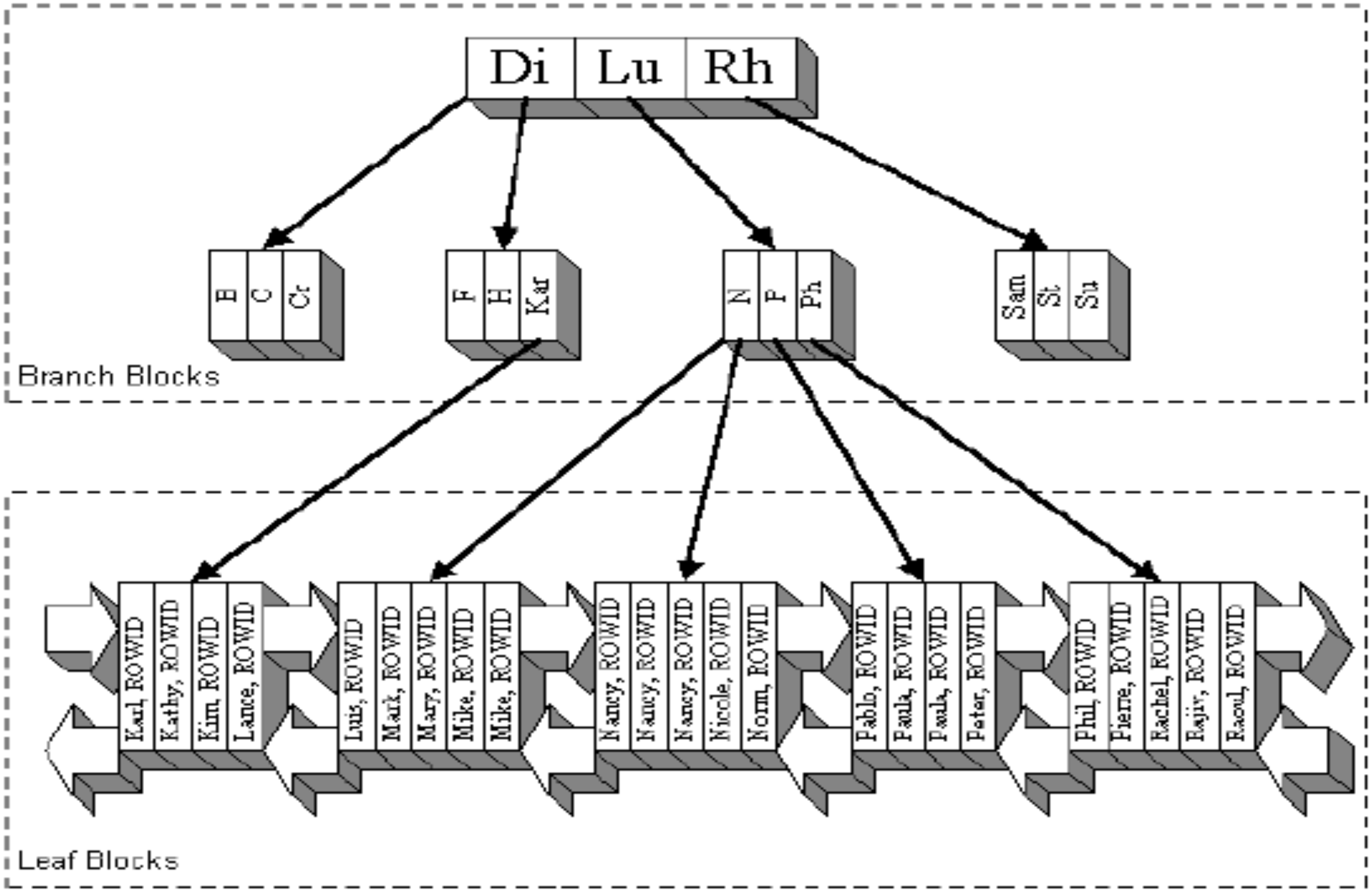
  SELECT * FROM emp WHERE UPPER(empname) = ’RICHARD’;

How Indexes Are Stored

- Oracle automatically allocates an index segment to hold the index’s data in a tablespace.
- The tablespace of an index’s segment is either the owner’s default tablespace or a tablespace specifically named in the CREATE INDEX statement.
- You can improve performance of queries that use an index by storing an index and its table in different tablespaces located on different disk drives.

Internal Structure of B-Tree Index

Figure 11–7 Internal Structure of a B-tree Index



Internal Structure of B-tree Index(contd.)

- The upper blocks (**branch blocks**) of a B-tree index contain index data that points to lower-level index blocks. The lowest level index blocks (**leaf blocks**) contain every indexed data value and a corresponding rowid used to locate the actual row.
- For a unique index, there is one rowid for each data value. For a nonunique index, the rowid is included in the key in sorted order, so nonunique indexes are sorted by the index key and rowid.

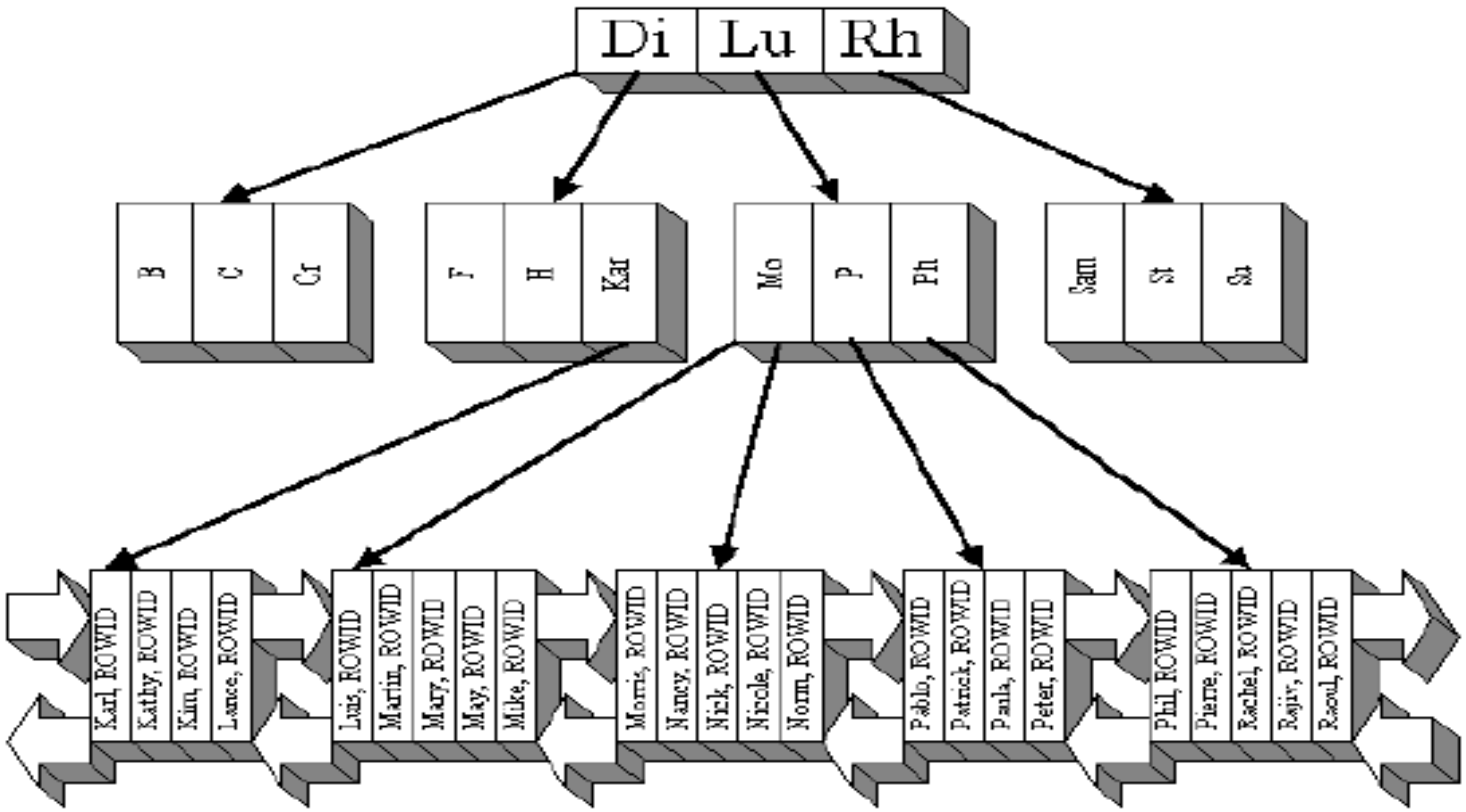
Advantages of B-Tree Structure

The B-tree structure has the following advantages:

- All leaf blocks of the tree are at the same depth, so retrieval of any record from anywhere in the index takes approximately the same amount of time.
- B-tree performance is good for both small and large tables and does not degrade as the size of a table grows.
- B-trees provide excellent retrieval performance for a wide range of queries,including exact match and range searches.
- Inserts, updates, and deletes are efficient, maintaining key order for fast retrieval.

Index Unique Scan

Figure 11–8 Example of an Index Unique Scan



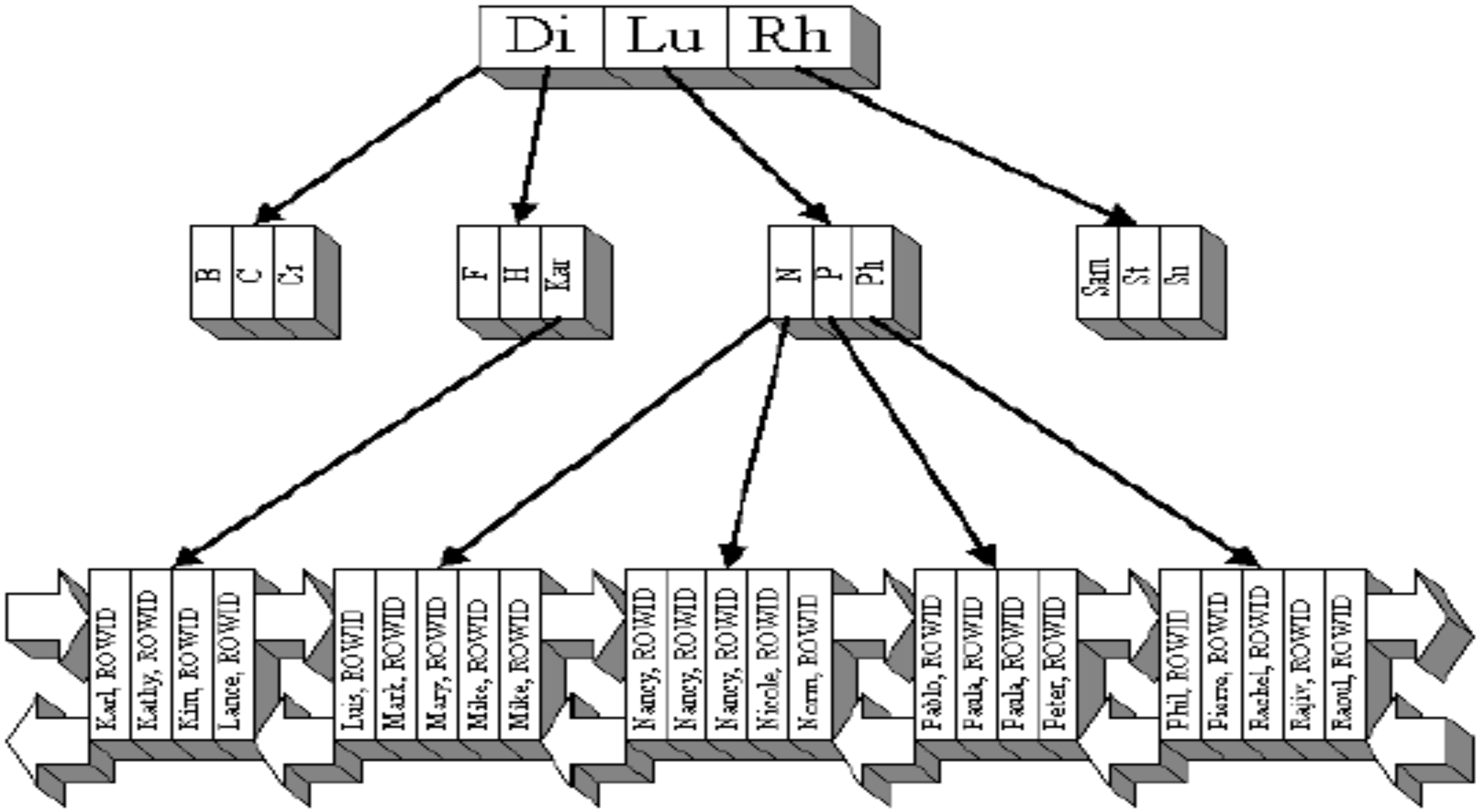
Index Unique Scan(contd.)

Eg: If searching for Patrick:

- In the root block, Rh is the smallest key >= Patrick.
- Follow the link before Rh to branch block (N, P, Ph).
- In this block, Ph is the smallest key >= Patrick.
- Follow the link before Ph to leaf block (Pablo, Patrick, Paula, Peter).
- In this block, search for key Patrick = Patrick.
- Found Patrick = Patrick, return `(KEY, ROWID)`.

Bounded Range Scan

Figure 11–9 Example of a Bounded Range Scan

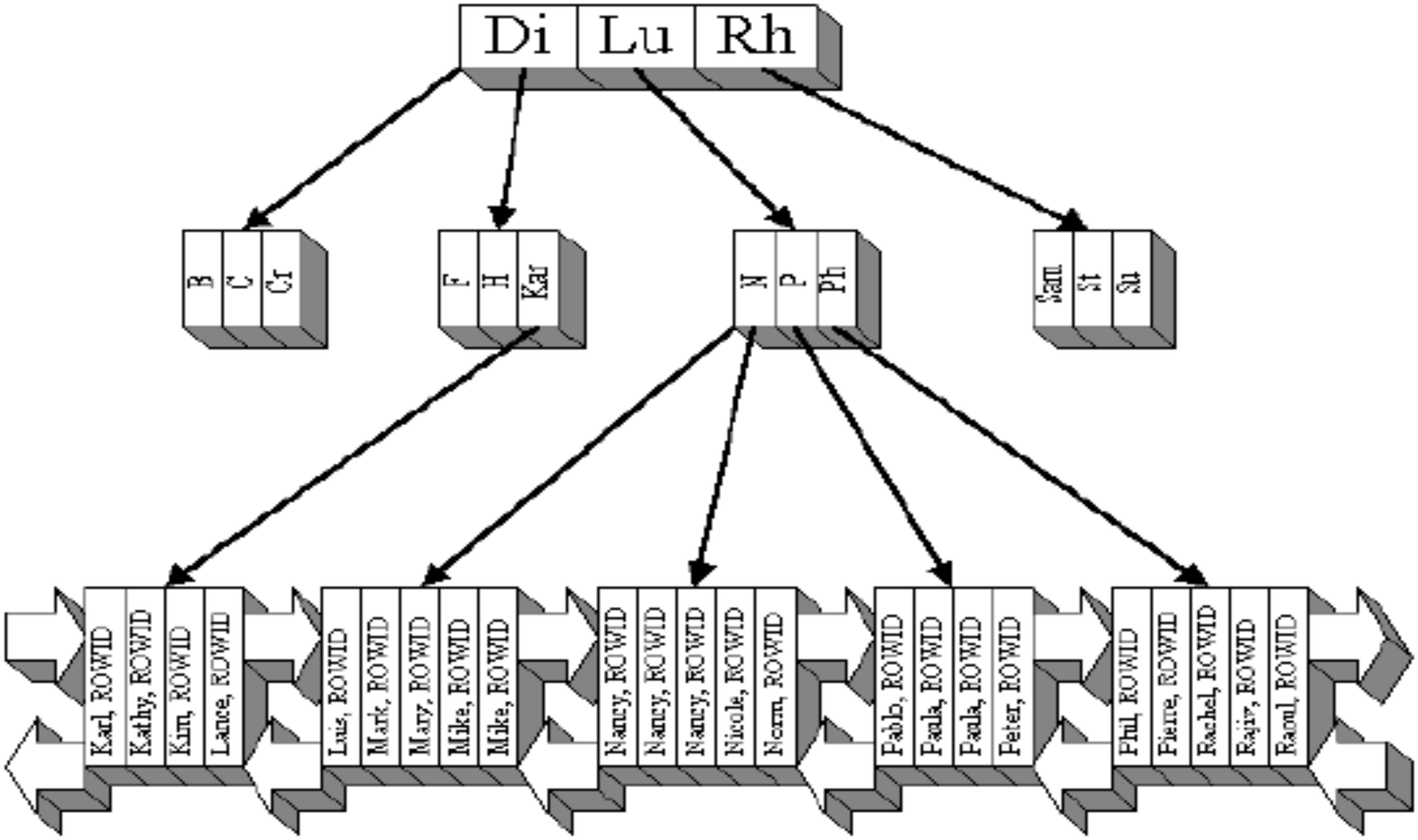


Bounded Range Scan(contd.)

With range scans using the non-unique B-tree index, if searching for Nancy:

- Start key = ‘Nancy’, end key < ‘Nancy’.
- In the root block, Rh is the smallest key >= start key.
- Follow the link before Rh to branch block (N, P, Ph).
- In this block, P is the smallest key >= start key.
- Follow the link before P to leaf block (Nancy, …, Nicole, Norm).
- In this block, Nancy is the smallest key >= start key.
- Because Nancy <= end key, return the (`KEY`, `ROWID`).
- Next key Nancy <= end key, return the (`KEY`, `ROWID`).
- Next key Nancy <= end key, return the (`KEY`, `ROWID`).
- Next key Nicole > end key, terminate the range scan.

Index Range Scan Descending



Index Range Scan Descending

If searching for ‘P%’:

- Start key = ‘P’, end key < ‘Q’.
- In the root block key, Lu is the biggest key <= end key.
- Follow the link to branch block (N, P, Ph).
- In this branch block, Ph is the biggest key <= end key.
- Follow the link to leaf block (Phil,…,Raoul).
- In the leaf block, Pierre is the biggest key <= end key.
- Pierre >= start key, return the (`KEY`, `ROWID`).
- Prev key Phil >= start key, return the (`KEY`, `ROWID`).
- Prev key Peter >= start key, return the (`KEY`, `ROWID`).
- Prev key Paula >= start key, return the (`KEY`, `ROWID`).
- Prev key Pablo >= start key, return the (`KEY`, `ROWID`).
- Prev key Norm < start key, terminate the range scan.

Key Compression

- Key compression lets you compress portions of the primary key column values in an index or index-organized table, which reduces the storage overhead of repeated values.

Prefix and Suffix Entries

- Key compression breaks the index key into a prefix entry (the grouping piece) and a suffix entry (the unique piece). Compression is achieved by sharing the prefix entries among the suffix entries in an index block.

Uses of Key Compression

- In a nonunique regular index, Oracle stores duplicate keys with the rowid appended to the key to break the duplicate rows. If key compression is used,Oracle stores the duplicate key as a prefix entry on the index block without the rowid. The rest of the rows are suffix entries that consist of only the rowid.
- This same behavior can be seen in a unique index that has a key of the form (**item**, **time stamp**), for example`(stock_ticker,transaction_time).`

  Thousands of rows can have the same `stock_ticker` value, with `transaction_time` preserving uniqueness. On a particular index block a `stock_ticker` value is stored only once as a prefix entry. Other entries on the index block are `transaction_time` values stored as suffix entries that reference the common `stock_ticker` prefix entry.

Reverse Key Indexes

- In reverse key index,the bytes of each column indexed (except the rowid) is reversed while keeping the column order.
- Such an arrangement can help avoid performance degradation with Oracle9*i* Real Application Clusters where modifications to the index are concentrated on a small set of leaf blocks.
- You can specify the keyword `REVERSE` along with the optional index specifications in a `CREATE INDEX` statement:

```
   CREATE INDEX i ON t (a,b,c)
REVERSE;
```

Bitmap Indexes

- In a **bitmap index**, a bitmap for each key value is used instead of a list of rowids.
- Each bit in the bitmap corresponds to a possible rowid. If the bit is set, then it means that the row with the corresponding rowid contains the key value.
- A mapping function converts the bit position to an actual rowid, so the bitmap index provides the same functionality as a regular index even though it uses a different representation internally.
- If the number of different key values is small, then bitmap indexes are very space efficient.

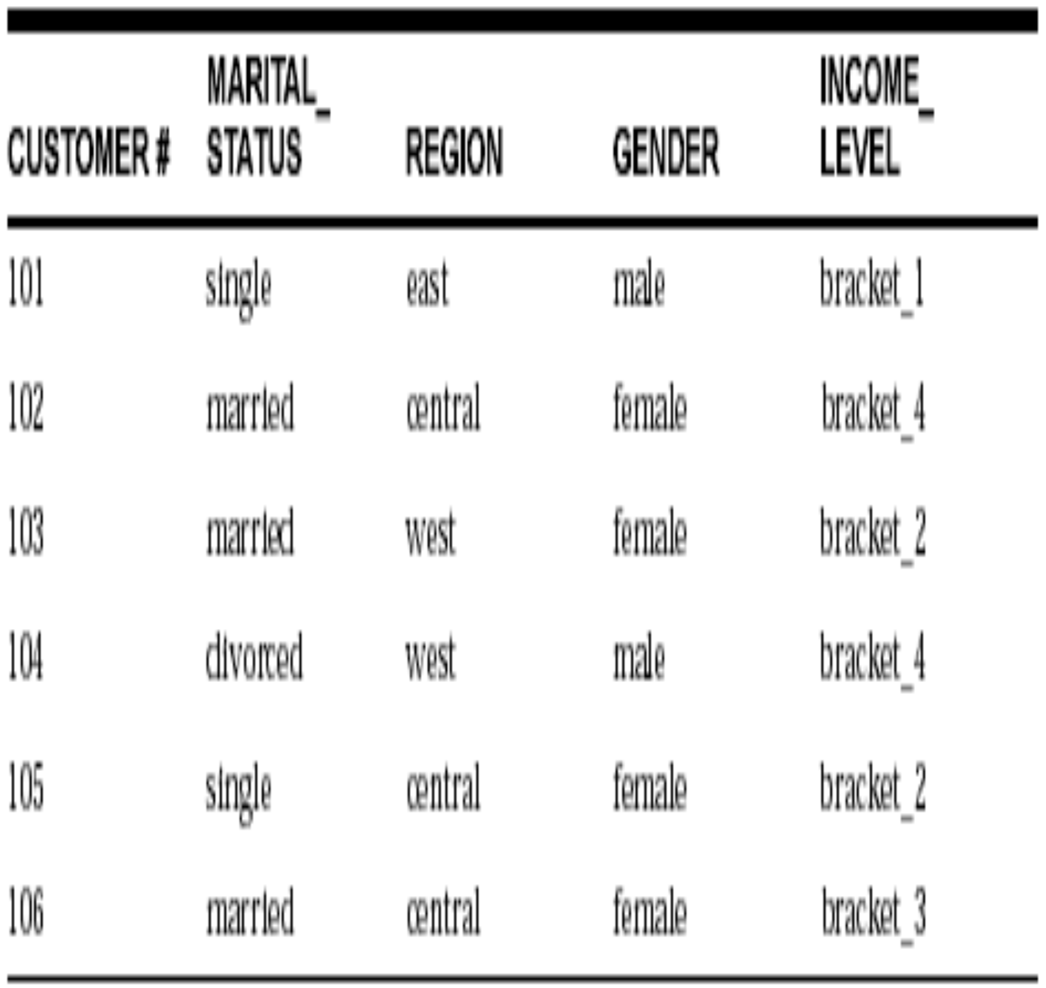
Bitmap Index Example

- MARITAL_STATUS, REGION, GENDER, and INCOME_LEVEL are all low-cardinality columns.

- **Cardinality** :

  The advantages of using bitmap indexes are greatest for low cardinality columns: that is, columns in which the number of distinct values is small compared to the number of rows in the table. If the number of distinct values of a column is less than 1% of the number of rows in the table, or if the values in a column are repeated more than 100 times, then the column is a candidate for a bitmap index.

Table 11-1 Bitmap Index Example

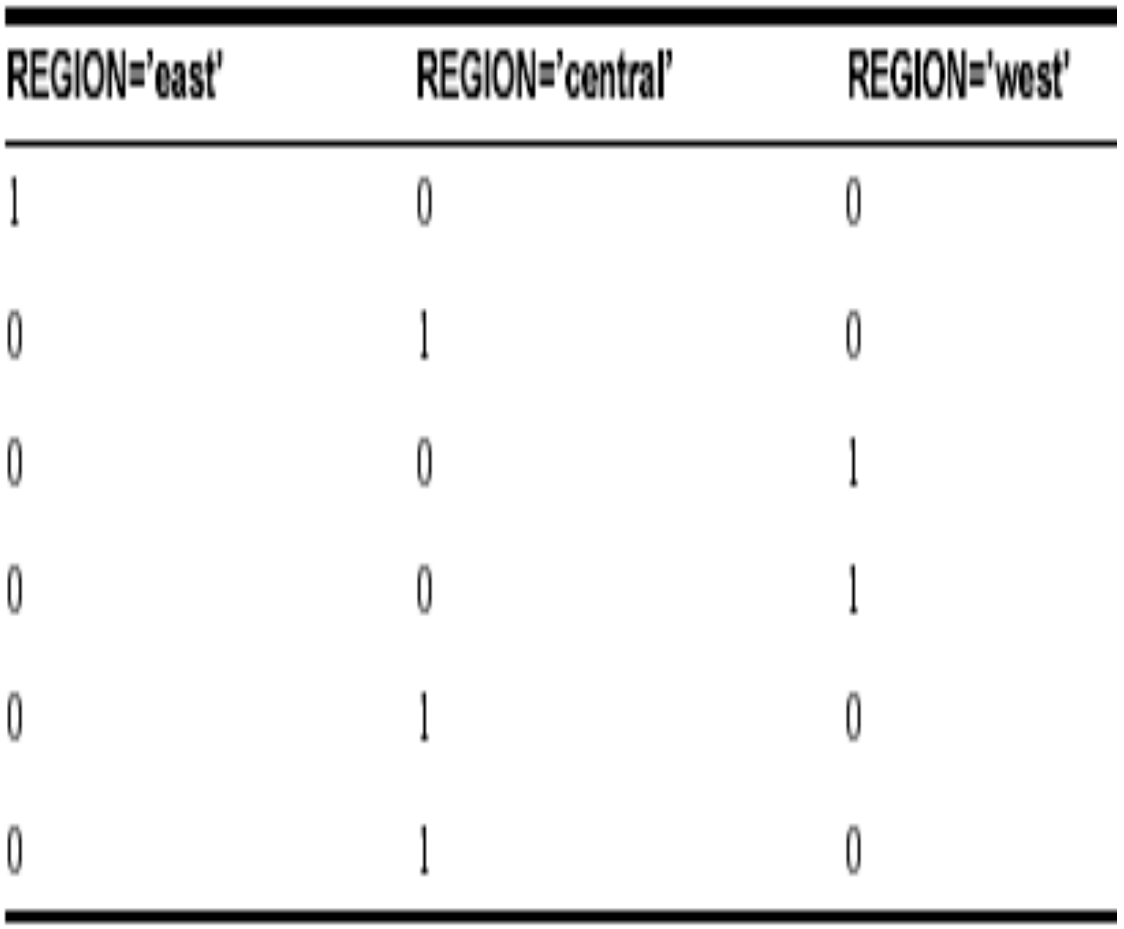


CUSTOMER #	MARITAL_ STATUS	REGION	GENDER	INCOME_ LEVEL
101	single	east	male	bracket_1
102	married	central	female	bracket_4
103	married	west	female	bracket_2
104	divorced	west	male	bracket_4
105	single	central	female	bracket_2
106	married	central	female	bracket_3

Bitmap Index Example(contd.)

- Table 11–2 illustrates the bitmap index for the REGION column in this example. It consists of three separate bitmaps, one for each region. Each entry or bit in the bitmap corresponds to a single row of the CUSTOMER table.
- The value of each bit depends upon the values of the corresponding row in the table. For instance, the bitmap REGION='east' contains a one as its first bit.

Table 11–2 Sample Bitmap



REGION='east'	REGION='central'	REGION='west'
1	0	0
0	1	0
0	0	1
0	0	1
0	1	0
0	1	0

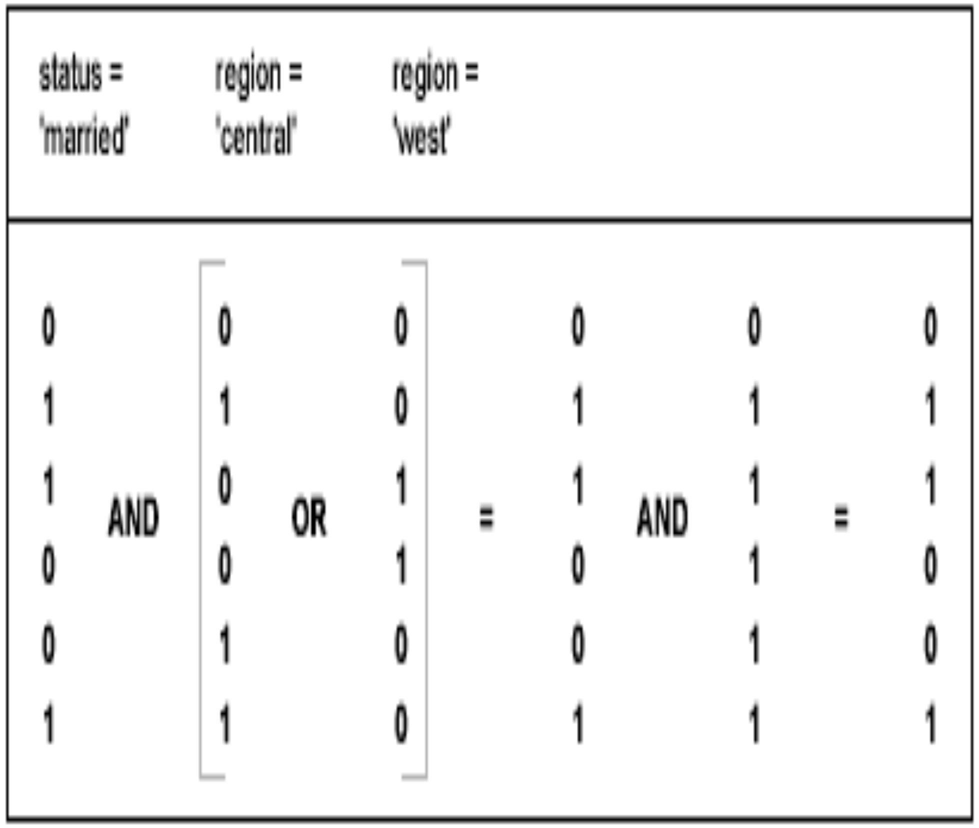
An analyst investigating demographic trends of the company's customers can ask,

"How many of our married customers live in the central or west regions?"

This corresponds to the following SQL query:

SELECT COUNT(*) FROM CUSTOMER WHERE MARITAL_STATUS = 'married' AND REGION IN ('central','west');

Figure 11–11 Executing a Query Using Bitmap Indexes



$$\begin{array}{c} 0\\1\\1\\0\\0\\1 \end{array} \text{ AND } \left[\begin{array}{c} 0\\1\\0\\0\\1\\1 \end{array} \text{ OR } \begin{array}{c} 0\\0\\1\\1\\0\\0 \end{array}\right] = \begin{array}{c} 0\\1\\1\\0\\0\\1 \end{array} \text{ AND } \begin{array}{c} 0\\1\\1\\1\\1\\1 \end{array} = \begin{array}{c} 0\\1\\1\\0\\0\\1 \end{array}$$

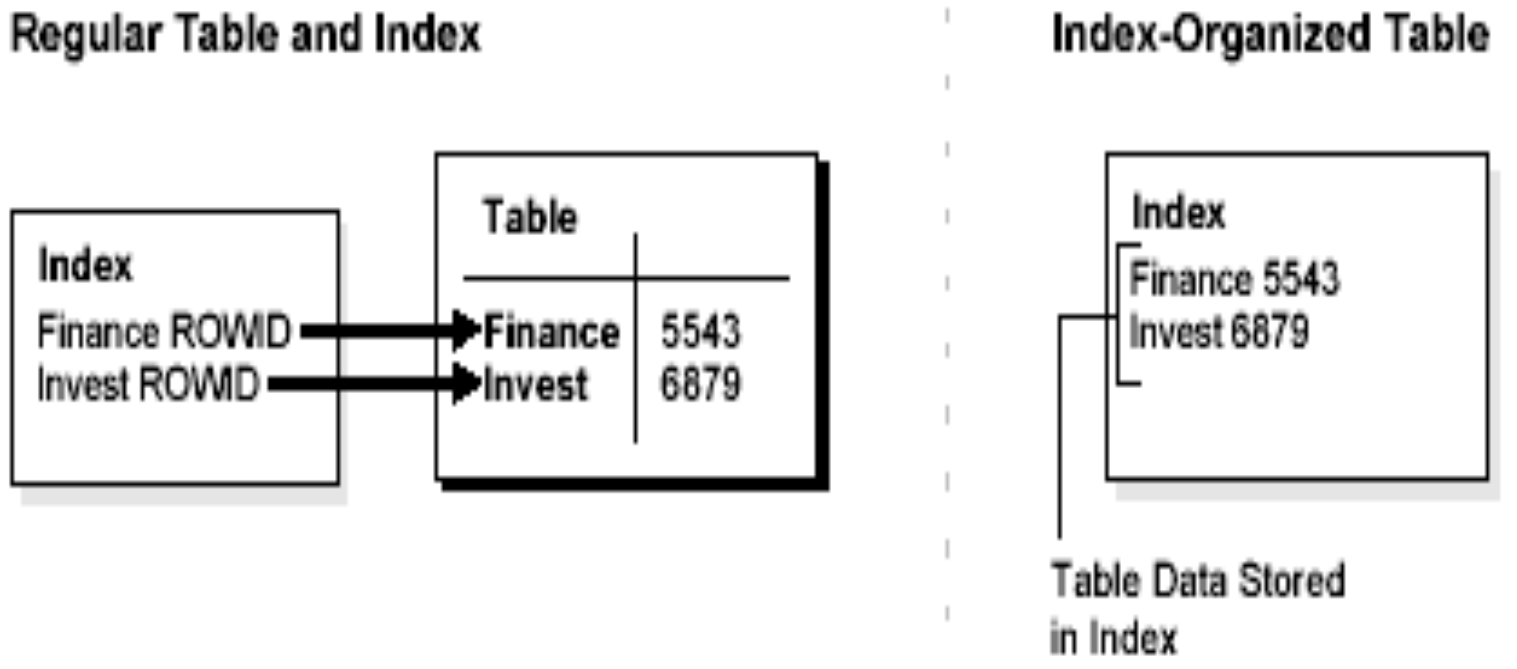
Bitmap Join Indexes

- A join index is an index on one table that involves columns of one or more different tables through a join.
- The bitmap join index, in its simplest form, is a bitmap index on a table `F` based on columns from table `D1,...,Dn,` where `Di` joins with `F` in a star or snowflake schema.
- We call the table whose rowids are bitmapped the ***fact table***, and the other tables participating in the join of bitmap join index the ***dimension tables***.

Index-Organized Tables

- An **index-organized table** has a storage organization that is a variant of a primary B-tree.
- The index-organized table is somewhat similar to a configuration consisting of an ordinary table and an index on one or more of the table columns, but instead of maintaining two separate storage structures, one for the table and one for the B-tree index, the database system maintains only a single B-tree index.

Figure 11–16 Structure of a Regular Table Compared with an Index-Organized Table



Application Domain Indexes

- Oracle provides **extensible indexing** to accommodate indexes on customized complex data types such as documents, spatial data, images, and video clips and to make use of specialized indexing techniques.

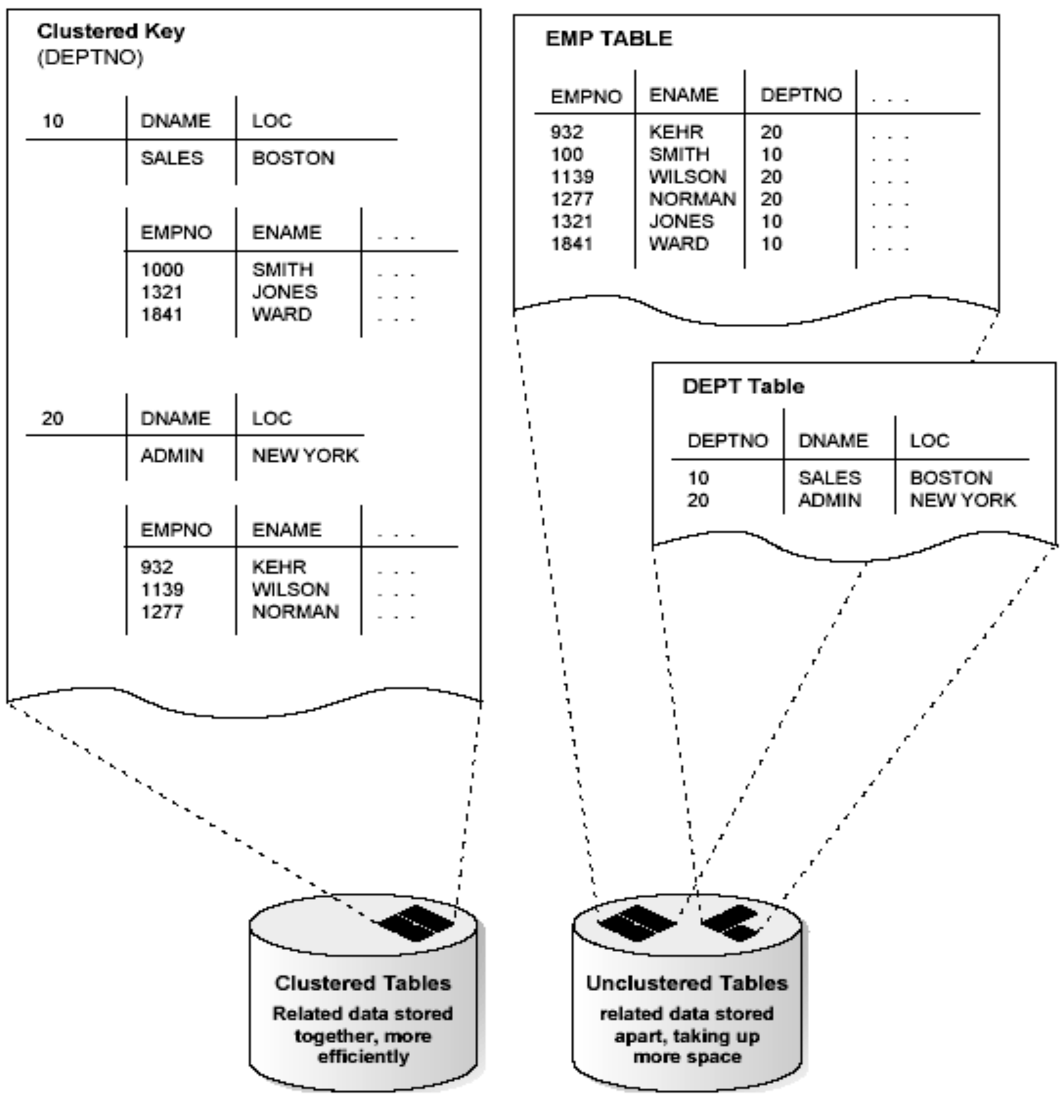
- With extensible indexing, you can encapsulate application-specific index management routines as an **indextype** schema object and define a **domain index** (an application-specific index) on table columns or attributes of an object type.

Clusters

- **Clusters** are an optional method of storing table data. A cluster is a group of tables that share the same data blocks because they share common columns and are often used together.

  For example, the EMP and DEPT table share the DEPTNO column. When you cluster the EMP and DEPT tables, Oracle physically stores all rows for each department from both the EMP and DEPT tables in the same data blocks.
- Figure 11–17 shows what happens when you cluster the EMP and DEPT tables:

Figure 11–17 Clustered Table Data



Hash Cluster

- Hashing is an optional way of storing table data to improve the performance of data retrieval.
- To use hashing, create a **hash cluster** and load tables into the cluster.
- Oracle physically stores the rows of a table in a hash cluster and retrieves them according to the results of a hash function.
- Oracle uses a **hash function** to generate a distribution of numeric values, called **hash values.**
- To find or store a row in a hash cluster, Oracle applies the hash function to the row’s cluster key value.

Hash Cluster(contd.)

- To find or store a row in an indexed table or cluster, at least two I/Os must be performed:
  - One or more I/Os to find or store the key value in the index
  - Another I/O to read or write the row in the table or cluster
- In contrast, Oracle uses a hash function to locate a row in a hash cluster. No I/O is required. As a result, a minimum of one I/O operation is necessary to read or write a row in a hash cluster.

SQL, PL/SQL,

Introduction to SQL

- SQL is a database access, nonprocedural language.
- Users describe in SQL what they want done, and the SQL language compiler automatically generates a procedure to navigate the database and perform the desired task.

SQL Statements

- A SQL statement is a computer program or instruction.
- A statement consists partially of SQL ***reserved words***.
- Oracle SQL statements are divided into the following categories:
    - Data manipulation language (DML) statements
    - Data definition language statements (DDL)
    - Transaction control statements
    - Session control statements
    - System control statements
    - Embedded SQL statements

Data Manipulation Language Statements

- Data manipulation language (DML) statements query or manipulate data in existing schema objects.
- DML enables one to:
  - Retrieve data from one or more tables or views *(SELECT)*
  - Add new rows of data into a table or view *(INSERT)*
  - Change column values in existing rows of a table or view *(UPDATE)*
  - Update or insert rows conditionally into a table or view *(MERGE)*
  - Remove rows from tables or views *(DELETE)*
  - See the execution plan for a SQL statement *(EXPLAIN PLAN)*
  - Lock a table or view, temporarily limiting other users' access *(LOCK TABLE)*

Data Definition Language Statements

- Data definition language (DDL) statements define, alter the structure of, and drop schema objects.
- DDL statements enable you to:
  - Create, alter, and drop schema objects and other database structures, including the database itself and database users *`(CREATE, ALTER, DROP)`*
  - Change the names of schema objects *`(RENAME)`*
  - Delete all the data in schema objects without removing the objects' structure *`(TRUNCATE)`*
  - Grant and revoke privileges and roles *`(GRANT, REVOKE)`*
  - Turn auditing options on and off *`(AUDIT, NOAUDIT)`*
  - Add a comment to the data dictionary *`(COMMENT)`*

Transaction Control Statements

- Transaction control statements manage the changes made by DML statements and group DML statements into transactions.
- They enable you to:
  - Make a transaction’s changes permanent *(COMMIT)*
  - Undo the changes in a transaction, either since the transaction started or since a savepoint *(ROLLBACK)*
  - Set a point to which you can roll back *(SAVEPOINT)*
  - Establish properties for a transaction *(SET TRANSACTION)*

Session Control Statements

- Session control statements manage the properties of a particular user's session.
- They enable you to:
  - Alter the current session by performing a specialized function, such as enabling and disabling the SQL trace facility *`(ALTER SESSION)`*
  - Enable and disable roles (groups of privileges) for the current session *`(SET ROLE)`*

System Control Statements

- System control statements change the properties of the Oracle server instance.
- The only system control statement is `ALTER SYSTEM.` It enables you to change settings (such as the minimum number of shared servers), kill a session, and perform other tasks.

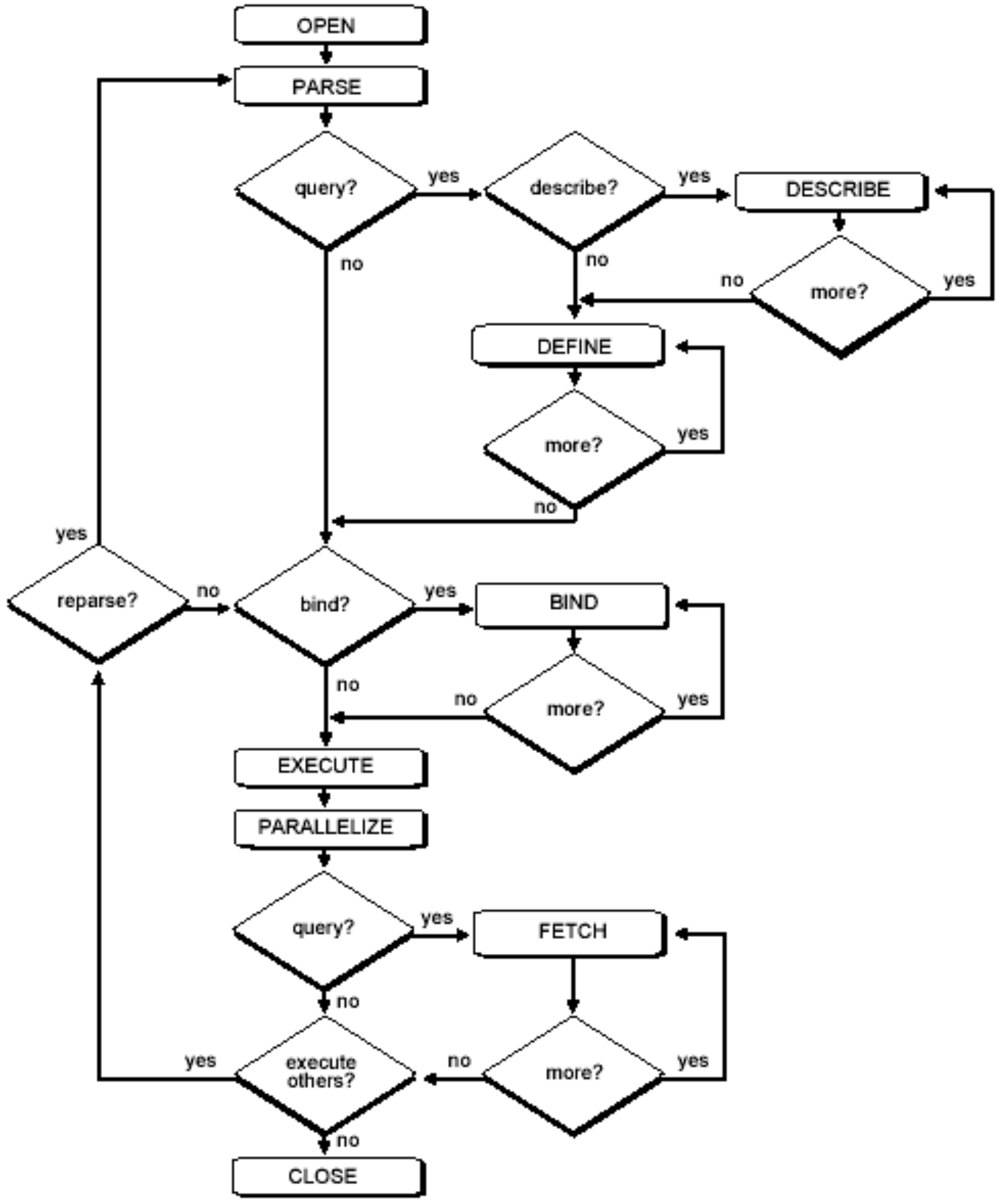
Embedded SQL Statements

- Embedded SQL statements incorporate DDL, DML, and transaction control statements within a procedural language program.
- They are used with the Oracle precompilers.
- Embedded SQL statements enable you to:
  - Define, allocate, and release cursors *(DECLARE CURSOR, OPEN, CLOSE)*
  - Specify a database and connect to Oracle *(DECLARE DATABASE, CONNECT)*
  - Assign variable names *(DECLARE STATEMENT)*
  - Initialize descriptors *(DESCRIBE)*
  - Specify how error and warning conditions are handled *(WHENEVER)*
  - Parse and execute SQL statements *(PREPARE, EXECUTE, EXECUTE IMMEDIATE)*
  - Retrieve data from the database *(FETCH)*

Cursors

- A ***cursor*** is a handle or name for a ***private SQL area*** - an area in memory in which a parsed statement and other information for processing the statement are kept.
- In application development, a cursor is a named resource available to a program and can be used specifically to parse SQL statements embedded within the application.
- ***Scrollable cursors*** are cursors in which fetches and DML operations do not need to be forward sequential only. Interfaces exist to fetch previously fetched rows, to fetch the nth row in the result set, and to fetch the nth row from the current position in the result set.

SQL Statement Execution



Macneil Fernandes©2005

DML Statement Processing

The following stages are necessary for each type of statement processing:

Stage 1: Create a Cursor

Stage 2: Parse the Statement

Stage 5: Bind Any Variables

Stage 7: Execute the Statement

Stage 9: Close the Cursor

Optionally, you can include another stage:

Stage 6: Parallelize the Statement

Queries `(SELECTs)` require several additional stages

Stage 3: Describe Results of a Query

Stage 4: Define Output of a Query

Stage 8: Fetch Rows of a Query

Stage 1: Create a Cursor

- A program interface call creates a cursor.
- The cursor is created independent of any SQL statement: it is created in expectation of any SQL statement.
- In most applications, cursor creation is automatic. However, in precompiler programs, cursor creation can either occur implicitly or be explicitly declared.

Stage 2: Parse the Statement

•During parsing, the SQL statement is passed from the user process to Oracle, and a parsed representation of the SQL statement is loaded into a shared SQL area.

•Parsing is the process of:

- Translating a SQL statement, verifying it to be a *valid statement*
- Performing data dictionary lookups to check table and column definitions
- Acquiring parse locks on required objects so that their definitions do not change during the statement's parsing
- Checking privileges to access referenced schema objects
- Determining the optimal execution plan for the statement
- Loading it into a shared SQL area
- Routing all or part of distributed statements to remote nodes that contain referenced data

Stage 3: Describe Results of a Query

- The describe stage is necessary only if the characteristics of a query’s result are not known; for example, when a query is entered interactively by a user.
- In this case, the describe stage determines the characteristics (datatypes, lengths, and names) of a query’s result.

Stage 4: Define Output of a Query

•In the define stage for queries, you specify the location, size, and datatype of variables defined to receive each fetched value. Oracle performs datatype conversion if necessary.

Stage 5: Bind Any Variables

•At this point, Oracle knows the meaning of the SQL statement but still does not have enough information to execute the statement. Oracle needs values for any variables listed in the statement.

•*A* program must specify the location (memory address) where the value can be found.

•A datatype and length for each value (unless they are implied or defaulted) should be specified if Oracle needs to perform datatype conversion.

Stage 6: Parallelize the Statement

- Parallelization causes multiple server processes to perform the work of the SQL statement so it can complete faster.
- Oracle can parallelize queries (`SELECT`s, `INSERT`s, `UPDATE`s, `MERGE`s, `DELETE`s), and some DDL operations such as index creation, creating a table with a subquery, and operations on partitions.

Stage 7: Execute the Statement

•Oracle has all necessary information and resources, so the statement is executed.

• If the statement is a query or an `INSERT` statement, no rows need to be locked because no data is being changed.

• If the statement is an `UPDATE` or `DELETE` statement, however, all rows that the statement affects are locked from use by other users of the database until the next `COMMIT, ROLLBACK,` or `SAVEPOINT` for the transaction. This ensures data integrity.

Stage 8: Fetch Rows of a Query

- In the fetch stage, rows are selected and ordered (if requested by the query), and each successive fetch retrieves another row of the result until the last row has been fetched.

Stage 9: Close the Cursor

- The final stage of processing a SQL statement is closing the cursor.

The Optimizer

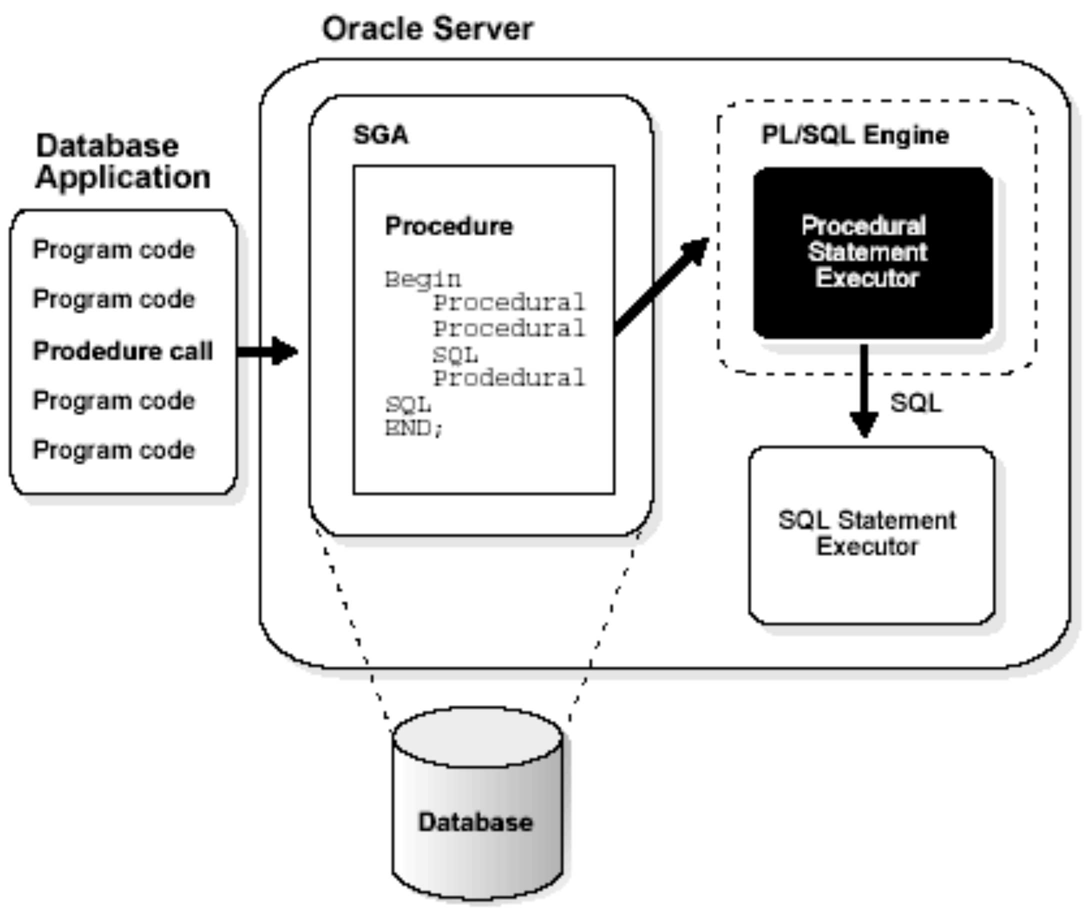
- The optimizer determines the most efficient way to execute a SQL statement.
- There are often many different ways to execute a SQL statement; for example, by varying the order in which tables or indexes are accessed. The procedure Oracle uses to execute a statement can greatly affect how quickly the statement executes.
- The optimizer considers many factors among alternative access paths. It can use either a cost-based or a rule-based approach.

PL/SQL

- PL/SQL is Oracle's procedural language extension to SQL.
- PL/SQL enables you to mix SQL statements with procedural constructs.
- PL/SQL program units generally are categorized as anonymous blocks and stored procedures.
    - An **anonymous block** is a PL/SQL block that appears within your application and it is not named or stored in the database.
    - A **stored procedure** is a PL/SQL block that Oracle stores in the database and can be called by name from an application.

PL/SQL Engine

- The program unit is stored in a database.
- When an application calls a procedure stored in the database, Oracle loads the compiled program unit into the shared pool in the system global area (SGA).
- The PL/SQL and SQL statement executors work together to process the statements within the procedure.



Language Constructs for PL/SQL

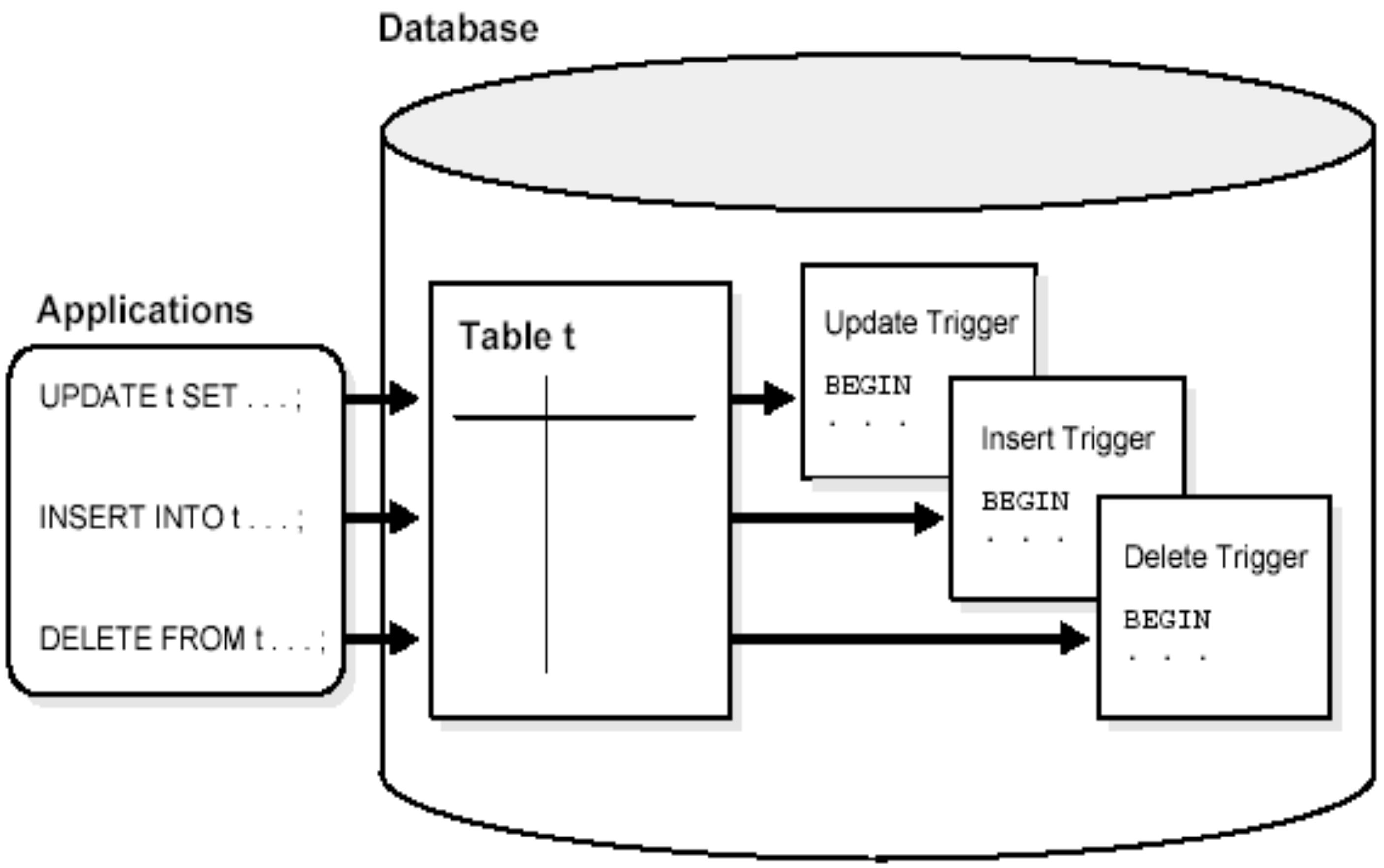
- PL/SQL blocks can include the following PL/SQL language constructs:
  - Variables and constants
  - Cursors
  - Exceptions
- **Stored Procedures -** Oracle also allows you to create and call stored procedures. If application calls a stored procedure, the parsed representation of the procedure is retrieved from the database and processed by the PL/SQL engine in Oracle.

Triggers

Introduction to Triggers

- Oracle allows you to define procedures called *triggers* that execute implicitly when an `INSERT, UPDATE,` or `DELETE` statement is issued against the associated table or, in some cases, against a view, or when database system actions occur.
- These procedures can be written in PL/SQL or Java and stored in the database, or they can be written as C callouts.
- Procedures and triggers differ in the way that they are invoked. A procedure is explicitly executed by a user, application, or trigger. Triggers are implicitly fired by Oracle when a triggering event occurs, no matter which user is connected or which application is being used.

Triggers



Database
Applications
UPDATE t SET . . . ;
INSERT INTO t . . . ;
DELETE FROM t . . . ;
Table t
Update Trigger
BEGIN
. . .
Insert Trigger
BEGIN
. . .
Delete Trigger
BEGIN
. . .

Introduction to Triggers

- The events that fire a trigger include the following:
  - DML statements that modify data in a table `(INSERT, UPDATE, or DELETE)`
  - DDL statements
  - System events such as startup, shutdown, and error messages
  - User events such as logon and logoff
- You can have multiple triggers of the same type for the same statement for any given table.

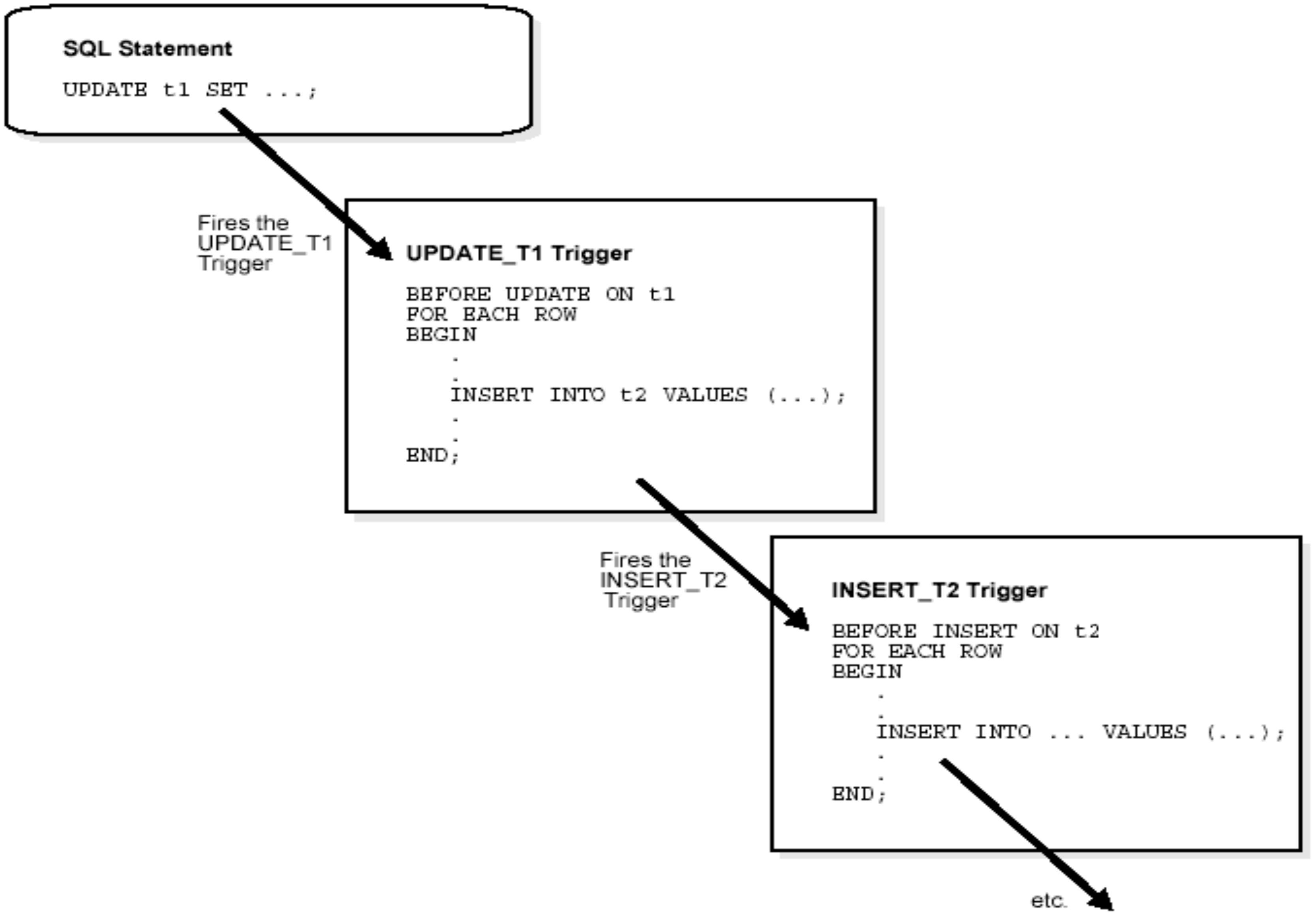
Triggers can be used for:

- Automatically generate derived column values
- Prevent invalid transactions
- Enforce complex security authorizations
- Enforce referential integrity across nodes in a distributed database
- Enforce complex business rules
- Provide transparent event logging
- Provide auditing
- Maintain synchronous table replicates
- Gather statistics on table access

Cascading Triggers

- When a trigger fires, a SQL statement within its trigger action potentially can fire other triggers, resulting in *cascading triggers.*
- These can produce unintended effects.

Cascading Triggers Contd…



SQL Statement

```
UPDATE t1 SET ...;
```

UPDATE_T1 Trigger

```
BEFORE UPDATE ON t1
FOR EACH ROW
BEGIN
   .
   .
   INSERT INTO t2 VALUES (...);
   .
   .
END;
```

INSERT_T2 Trigger

```
BEFORE INSERT ON t2
FOR EACH ROW
BEGIN
   .
   .
   INSERT INTO ... VALUES (...);
   .
   .
END;
```

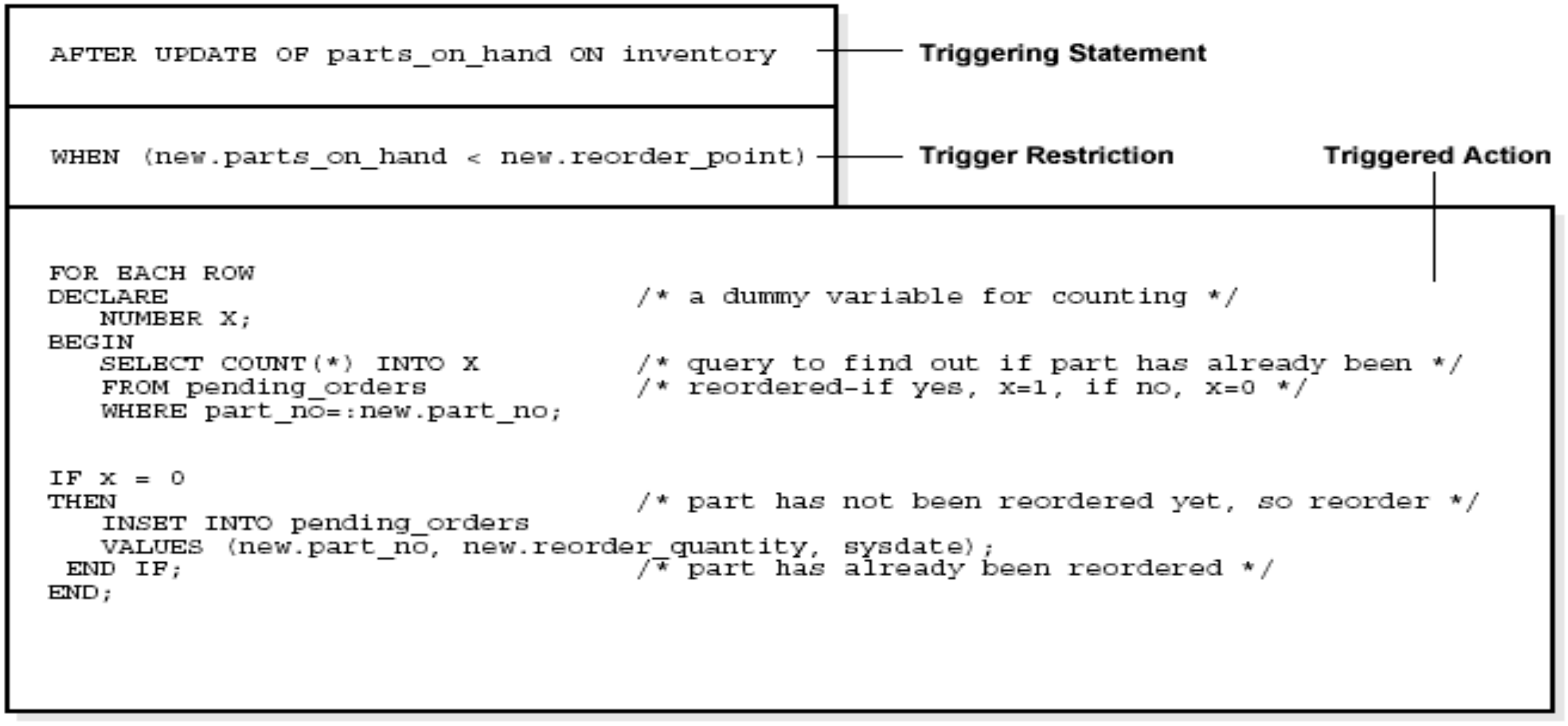
Triggers compared with Declarative Integrity Constraints

- You can use both triggers and integrity constraints to define and enforce any type of integrity rule.
- It is recommended, that you use triggers to constrain data input only in the following situations:
  - To enforce referential integrity when child and parent tables are on different nodes of a distributed database.
  - To enforce complex business rules not definable using integrity constraints
  - When a required referential integrity rule cannot be enforced using the following integrity constraints:

```
NOT NULL, UNIQUE
PRIMARY KEY
FOREIGN KEY
CHECK
DELETE CASCADE
DELETE SET NULL
```

Parts of a Trigger

- A trigger has three basic parts:
  - A triggering event or statement
  - A trigger restriction
  - A trigger action



```
AFTER UPDATE OF parts_on_hand ON inventory
```

```
WHEN (new.parts_on_hand < new.reorder_point)
```

```
FOR EACH ROW
DECLARE                              /* a dummy variable for counting */
   NUMBER X;
BEGIN
   SELECT COUNT(*) INTO X            /* query to find out if part has already been */
   FROM pending_orders               /* reordered-if yes, x=1, if no, x=0 */
   WHERE part_no=:new.part_no;


IF x = 0
THEN                                 /* part has not been reordered yet, so reorder */
   INSET INTO pending_orders
   VALUES (new.part_no, new.reorder_quantity, sysdate);
 END IF;                             /* part has already been reordered */
END;
```

The Triggering Event or Statement

- A triggering event or statement is the SQL statement, database event, or user event that causes a trigger to fire.
- A triggering event can be one or more of the following:
  - An `INSERT, UPDATE,` or `DELETE` statement on a specific table (or view, in some cases)
  - A `CREATE, ALTER,` or `DROP` statement on any schema object
  - A database startup or instance shutdown
  - A specific error message or any error message
  - A user logon or logoff
- When the triggering event is an `UPDATE` statement, you can include a column list to identify which columns must be updated to fire the trigger. You cannot specify a column list for `INSERT` and `DELETE` statements, because they affect entire rows of information.

Trigger Restriction

- A trigger restriction specifies a Boolean expression that must be `TRUE` for the trigger to fire.
- The trigger action is not executed if the trigger restriction evaluates to `FALSE` or `UNKNOWN.`

Trigger Action

- A trigger action is the procedure (PL/SQL block, Java program, or C callout) that contains the SQL statements and code to be executed when the following events occur:
  - A triggering statement is issued.
  - The trigger restriction evaluates to `TRUE.`
- Like stored procedures, a trigger action can:
  - Contain SQL, PL/SQL, or Java statements
  - Define PL/SQL language constructs such as variables, constants, cursors, exceptions
  - Define Java language constructs
  - Call stored procedures

Types of Triggers

- The different types of triggers:
  - Row Triggers and Statement Triggers
  - BEFORE and AFTER Triggers
  - INSTEAD OF Triggers
  - Triggers on System Events and User Events

Row Triggers and Statement Triggers

- When you define a trigger, you can specify the number of times the trigger action is to be executed:
  - Once for every row affected by the triggering statement, such as a trigger fired by an `UPDATE` statement that updates many rows
  - Once for the triggering statement, no matter how many rows it affects

Row Triggers and Statement Triggers Contd…

- A *row trigger* is fired each time the table is affected by the triggering statement. If an `UPDATE` statement updates multiple rows of a table, a row trigger is fired once for each row affected by the `UPDATE` statement. If a triggering statement affects no rows, a row trigger is not executed.
- A *statement trigger* is fired once on behalf of the triggering statement, regardless of the number of rows in the table that the triggering statement affects, even if no rows are affected.

BEFORE and AFTER Triggers

- When defining a trigger, you can specify the *trigger timing*—whether the trigger action is to be executed before or after the triggering statement. `BEFORE` and `AFTER` apply to both statement and row triggers.
- `BEFORE` and `AFTER` triggers fired by DML statements can be defined only on tables, not on views.
- `BEFORE` and `AFTER` triggers fired by DDL statements can be defined only on the database or a schema, not on particular tables.

INSTEAD OF Triggers

- `INSTEAD OF` triggers provide a transparent way of modifying views that cannot be modified directly through DML statements.
- These triggers are called `INSTEAD OF` triggers because, unlike other types of triggers, Oracle fires the trigger instead of executing the triggering statement.
- `INSTEAD OF` triggers are activated for each row of the view that gets modified.

Triggers on System Events and User Events

- System events
  - Database startup and shutdown
  - Server error message events
- User events
  - User logon and logoff
  - DDL statements (`CREATE`, `ALTER`, and `DROP`)
  - DML statements (`INSERT`, `DELETE`, and `UPDATE`)
- Triggers on system events can be defined at the database level or schema level.

Trigger Execution

Oracle automatically performs the following actions:

- Executes triggers of each type in a planned firing sequence when more than one trigger is fired by a single SQL statement
- Performs integrity constraint checking
- Provides read-consistent views for queries and constraints
- Manages the dependencies among triggers and schema objects referenced in the code of the trigger action
- Uses two-phase commit if a trigger updates remote tables in a distributed database
- Fires multiple triggers in an unspecified order, if more than one trigger of the same type exists for a given statement

The Execution Model for Triggers and Integrity Constraint Checking

- Oracle uses the following execution model to maintain the proper firing sequence of multiple triggers and constraint checking:
  - Execute all `BEFORE` *`statement`* triggers that apply to the statement.
  - Loop for each row affected by the SQL statement.
    - Execute all `BEFORE` *`row`* triggers that apply to the statement.
    - Lock and change row, and perform integrity constraint checking. (The lock is not released until the transaction is committed.)
    - Execute all `AFTER` *`row`* triggers that apply to the statement.
  - Complete deferred integrity constraint checking.
  - Execute all `AFTER` *`statement`* triggers that apply to the statement.

Data Access for Triggers

- The SQL statements executed within triggers follow the common rules used for standalone SQL statements.
- In particular, if an uncommitted transaction has modified values that a trigger being fired either needs to read (query) or write (update), the SQL statements in the body of the trigger being fired use the following guidelines:
  - Queries see the current read-consistent materialized view of referenced tables and any data changed within the same transaction.
  - Updates wait for existing data locks to be released before proceeding.

Storage of PL/SQL Triggers

- Oracle stores PL/SQL triggers in compiled form, just like stored procedures.
- When a `CREATE TRIGGER` statement commits, the compiled PL/SQL code, called P code (for pseudocode), is stored in the database and the source code of the trigger is flushed from the shared pool.

Execution of Triggers

- Oracle executes a trigger internally using the same steps used for procedure execution.
- The only subtle difference is that a user has the right to fire a trigger if he or she has the privilege to execute the triggering statement.

Dependency Maintenance for Triggers

- Like procedures, triggers depend on referenced objects.
- Oracle automatically manages the dependencies of a trigger on the schema objects referenced in its trigger action.
- The dependency issues for triggers are the same as those for stored procedures. Triggers are treated like stored procedures. They are inserted into the data dictionary.

THANK YOU