Introduction to Oracle Database

This chapter provides an overview of Oracle Database.

This chapter contains the following topics:

About Relational Databases

Every organization has information that it must store and manage to meet its requirements. For example, a corporation must collect and maintain human resources records for its employees. This information must be available to those who need it.

An information system is a formal system for storing and processing information. An information system could be a set of cardboard boxes containing manila folders along with rules for how to store and retrieve the folders. However, most companies today use a database to automate their information systems. A database is an organized collection of information treated as a unit. The purpose of a database is to collect, store, and retrieve related information for use by database applications.

Database Management System (DBMS)

A **database management system (DBMS)** is software that controls the storage, organization, and retrieval of data.

Typically, a DBMS has the following elements:

- Kernel code

  This code manages memory and storage for the DBMS.
- Repository of metadata

  This repository is usually called a data dictionary.
- Query language

  This language enables applications to access the data.

  A database application is a software program that interacts with a database to access and manipulate data.

The first generation of database management systems included the following types:

- Hierarchical

  A hierarchical database organizes data in a tree structure. Each parent record has one or more child records, similar to the structure of a file system.

- Network

  A network database is similar to a hierarchical database, except records have a many-to-many rather than a one-to-many relationship.

  The preceding database management systems stored data in rigid, predetermined relationships. Because no data definition language existed, changing the structure of the data was difficult. Also, these systems lacked a simple query language, which hindered application development.

Relational Model

In his seminal 1970 paper "A Relational Model of Data for Large Shared Data Banks," E. F. Codd defined a relational model based on mathematical set theory. Today, the most widely accepted database model is the relational model.

A relational database is a database that conforms to the relational model. The relational model has the following major aspects:

- Structures

  Well-defined objects store or access the data of a database.

- Operations

  Clearly defined actions enable applications to manipulate the data and structures of a database.

- Integrity rules

  Integrity rules govern operations on the data and structures of a database.

A relational database stores data in a set of simple relations. A relation is a set of tuples. A tuple is an unordered set of attribute values.
A table is a two-dimensional representation of a relation in the form of rows (tuples) and columns (attributes). Each row in a table has the same set of columns. A relational database is a database that stores data in relations (tables). For example, a relational database could store information about company employees in an employee table, a department table, and a salary table.

See also

"A Relational Model of Data for Large Shared Data Banks" for an abstract and link to Codd's paper

Relational Database Management System (RDBMS)

The relational model is the basis for a **relational database management system (RDBMS)**. An RDBMS moves data into a database, stores the data, and retrieves it so that applications can manipulate it.

An RDBMS distinguishes between the following types of operations:

- Logical operations

  In this case, an application specifies *what* content is required. For example, an application requests an employee name or adds an employee record to a table.

- Physical operations

In this case, the RDBMS determines *how* things should be done and carries out the operation. For example, after an application queries a table, the database may use an index to find the requested rows, read the data into memory, and perform many other steps before returning a result to the user. The RDBMS stores and retrieves data so that physical operations are transparent to database applications.
Oracle Database is an RDBMS. An RDBMS that implements object-oriented features such as user-defined types, inheritance, and polymorphism is called an object-relational database management system (ORDBMS). Oracle Database has extended the relational model to an object-relational model, making it possible to store complex business models in a relational database.

Brief History of Oracle Database

The current version of Oracle Database is the result of over 35 years of innovative development.

Highlights in the evolution of Oracle Database include the following:

- Founding of Oracle

  In 1977, Larry Ellison, Bob Miner, and Ed Oates started the consultancy Software Development Laboratories, which became Relational Software, Inc. (RSI). In 1983, RSI became Oracle Systems Corporation and then later Oracle Corporation.

- First commercially available RDBMS

  In 1979, RSI introduced Oracle V2 (Version 2) as the first commercially available SQL-based RDBMS, a landmark event in the history of relational databases.

- Portable version of Oracle Database

  Oracle Version 3, released in 1983, was the first relational database to run on mainframes, minicomputers, and PCs. The database was written in C, enabling the database to be ported to multiple platforms.

- Enhancements to concurrency control, data distribution, and scalability

  Version 4 introduced multiversion read consistency. Version 5, released in 1985, supported client/server computing and distributed database systems. Version 6 brought enhancements to disk I/O, row locking, scalability, and backup and recovery. Also, Version 6 introduced the first version of the PL/SQL language, a proprietary procedural extension to SQL.

- PL/SQL stored program units

  Oracle7, released in 1992, introduced PL/SQL stored procedures and triggers.

- Objects and partitioning

  Oracle8 was released in 1997 as the object-relational database, supporting many new data types. Additionally, Oracle8 supported partitioning of large tables.

- Internet computing

  Oracle8i Database, released in 1999, provided native support for internet protocols and server-side support for Java. Oracle8i was designed for internet computing, enabling the database to be deployed in a multitier environment.

- Oracle Real Application Clusters (Oracle RAC)

  Oracle9i Database introduced Oracle RAC in 2001, enabling multiple instances to access a single database simultaneously. Additionally, Oracle XML Database (Oracle XML DB) introduced the ability to store and query XML.
- Grid computing

  Oracle Database 10g introduced grid computing in 2003. This release enabled organizations to virtualize computing resources by building a grid infrastructure based on low-cost commodity servers. A key goal was to make the database self-managing and self-tuning. Oracle Automatic Storage Management (Oracle ASM) helped achieve this goal by virtualizing and simplifying database storage management.
- Manageability, diagnosability, and availability

  Oracle Database 11g, released in 2007, introduced a host of new features that enabled administrators and developers to adapt quickly to changing business requirements. The key to adaptability is simplifying the information infrastructure by consolidating information and using automation wherever possible.
- Plugging In to the Cloud

  Oracle Database 12c, released in 2013, was designed for the Cloud, featuring a new Multitenant architecture, In-Memory column store, and support for JSON documents. Oracle Database 12c helps customers make more efficient use of their IT resources, while continuing to reduce costs and improve service levels for users.

Schema Objects

One characteristic of an RDBMS is the independence of physical data storage from logical data structures.

In Oracle Database, a database schema is a collection of logical data structures, or schema objects. A database user owns a database schema, which has the same name as the user name.

Schema objects are user-created structures that directly refer to the data in the database. The database supports many types of schema objects, the most important of which are tables and indexes.

A schema object is one type of database object. Some database objects, such as profiles and roles, do not reside in schemas.

See also

"Introduction to Schema Objects" to learn more about schema object types, storage, and dependencies

Tables

A table describes an entity such as employees.
You define a table with a table name, such as `employees`, and set of columns. In general, you give each column a name, a data type, and a width when you create the table.
A table is a set of rows. A column identifies an attribute of the entity described by the table, whereas a row identifies an instance of the entity. For example, attributes of the

employees entity correspond to columns for employee ID and last name. A row identifies a specific employee.
You can optionally specify a rule, called an integrity constraint, for a column. One example is a `NOT NULL` integrity constraint. This constraint forces the column to contain a value in every row.

See also

- "Overview of Tables" to learn about columns and rows, data types, table storage, and table compression
- "Data Integrity" to learn about the possible types and states of constraints

Indexes

An **index** is an optional data structure that you can create on one or more columns of a table. Indexes can increase the performance of data retrieval.

When processing a request, the database can use available indexes to locate the requested rows efficiently. Indexes are useful when applications often query a specific row or range of rows.

Indexes are logically and physically independent of the data. Thus, you can drop and create indexes with no effect on the tables or other indexes. All applications continue to function after you drop an index.

See also

"Introduction to Indexes" to learn about the purpose and types of indexes

Data Access

A general requirement for a DBMS is to adhere to accepted industry standards for a data access language.

Structured Query Language (SQL)

SQL is a set-based declarative language that provides an interface to an RDBMS such as Oracle Database.

Procedural languages such as C describe *how* things should be done. SQL is nonprocedural and describes *what* should be done.

SQL is the ANSI standard language for relational databases. All operations on the data in an Oracle database are performed using SQL statements. For example, you use SQL to create tables and query and modify data in tables.

A SQL statement can be thought of as a very simple, but powerful, computer program or instruction. Users specify the result that they want (for example, the names of employees), not how to derive it. A SQL statement is a string of SQL text such as the following:

```
COPY
```

```
SELECT first_name, last_name FROM employees;
```

SQL statements enable you to perform the following tasks:

- Query data
- Insert, update, and delete rows in a table
- Create, replace, alter, and drop objects
- Control access to the database and its objects
- Guarantee database consistency and integrity

SQL unifies the preceding tasks in one consistent language. Oracle SQL is an implementation of the ANSI standard. Oracle SQL supports numerous features that extend beyond standard SQL.

See also

"SQL" to learn more about SQL standards and the main types of SQL statements

PL/SQL and Java

PL/SQL is a procedural extension to Oracle SQL.

PL/SQL is integrated with Oracle Database, enabling you to use all of the Oracle Database SQL statements, functions, and data types. You can use PL/SQL to control the flow of a SQL program, use variables, and write error-handling procedures.

A primary benefit of PL/SQL is the ability to store application logic in the database itself. A PL/SQL procedure or function is a schema object that consists of a set of SQL statements and other PL/SQL constructs, grouped together, stored in the database, and run as a unit to solve a specific problem or to perform a set of related tasks. The principal benefit of server-side programming is that built-in functionality can be deployed anywhere.

Oracle Database can also store program units written in Java. A Java stored procedure is a Java method published to SQL and stored in the database for general use. You can call existing PL/SQL programs from Java and Java programs from PL/SQL.

See also

- "Server-Side Programming: PL/SQL and Java"
- "Client-Side Database Programming"

Transaction Management

Oracle Database is designed as a multiuser database. The database must ensure that multiple users can work concurrently without corrupting one another's data.

Transactions

A **transaction** is a logical, atomic unit of work that contains one or more SQL statements.

An RDBMS must be able to group SQL statements so that they are either all committed, which means they are applied to the database, or all rolled back, which means they are undone.

An illustration of the need for transactions is a funds transfer from a savings account to a checking account. The transfer consists of the following separate operations:

1. Decrease the savings account.
2. Increase the checking account.
3. Record the transaction in the transaction journal.

Oracle Database guarantees that all three operations succeed or fail as a unit. For example, if a hardware failure prevents a statement in the transaction from executing, then the other statements must be rolled back.

Transactions are one feature that set Oracle Database apart from a file system. If you perform an atomic operation that updates several files, and if the system fails halfway through, then the files will not be consistent. In contrast, a transaction moves an Oracle database from one consistent state to another. The basic principle of a transaction is "all or nothing": an atomic operation succeeds or fails as a whole.

See also

"Transactions" to learn about the definition of a transaction, statement-level atomicity, and transaction control

Data Concurrency

A requirement of a multiuser RDBMS is the control of **data concurrency**, which is the simultaneous access of the same data by multiple users.
Without concurrency controls, users could change data improperly, compromising data integrity. For example, one user could update a row while a different user simultaneously updates it.

If multiple users access the same data, then one way of managing concurrency is to make users wait. However, the goal of a DBMS is to reduce wait time so it is either nonexistent or negligible. All SQL statements that modify data must proceed with as little interference as possible. Destructive interactions, which are interactions that incorrectly update data or alter underlying data structures, must be avoided.

Oracle Database uses locks to control concurrent access to data. A lock is a mechanism that prevents destructive interaction between transactions accessing a shared resource. Locks help ensure data integrity while allowing maximum concurrent access to data.

See also

"Overview of the Oracle Database Locking Mechanism"

Data Consistency

In Oracle Database, each user must see a consistent view of the data, including visible changes made by a user's own transactions and committed transactions of other users.

For example, the database must prevent the lost update problem, which occurs when one transaction sees uncommitted changes made by another concurrent transaction. Oracle Database always enforces statement-level read consistency, which guarantees that the data that a single query returns is committed and consistent for a single point in time. Depending on the transaction isolation level, this point is the time at which the statement was opened or the time the transaction began. The Oracle Flashback Query feature enables you to specify this point in time explicitly.

The database can also provide read consistency to all queries in a transaction, known as transaction-level read consistency. In this case, each statement in a transaction sees data from the same point in time, which is the time at which the transaction began.

See also

- "Data Concurrency and Consistency" to learn more about lost updates
- *Oracle Database Development Guide* to learn about Oracle Flashback Query

Oracle Database Architecture

A **database server** is the key to information management.
In general, a server reliably manages a large amount of data in a multiuser environment so that users can concurrently access the same data. A database server also prevents unauthorized access and provides efficient solutions for failure recovery.

Database and Instance

An Oracle database server consists of a database and at least one **database instance**, commonly referred to as simply an *instance*. Because an instance and a database are so closely connected, the term **Oracle database** is sometimes used to refer to both instance and database.

In the strictest sense the terms have the following meanings:

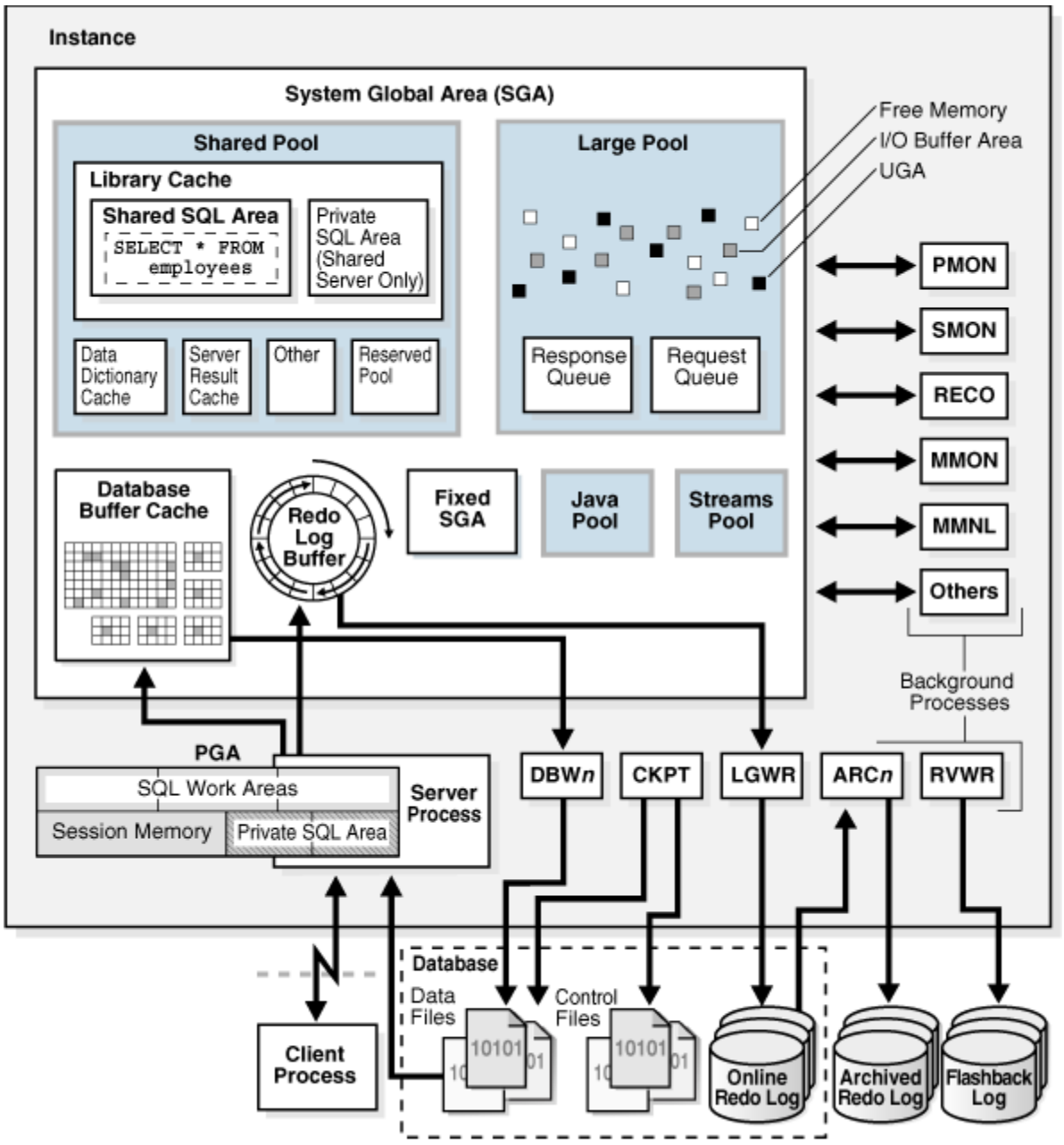
- Database

  A database is a set of files, located on disk, that store data. These files can exist independently of a database instance.

- Database instance

  An instance is a set of memory structures that manage database files. The instance consists of a shared memory area, called the system global area (SGA), and a set of background processes. An instance can exist independently of database files.
  The following figure shows a database and its instance. For each user connection to the instance, a client process runs the application. Each client process is associated with its own server process. The server process has its own private session memory, known as the program global area (PGA).

  Figure 1-1 Oracle Instance and Database



Description of "Figure 1-1 Oracle Instance and Database"

Although in the strict sense an Oracle database is a set of physical structures (files and memory structures), applications can interact with multiple logical databases inside a single physical database, or a single logical database distributed across multiple physical databases.

This section contains the following topics:

- Multitenant Architecture
- Sharding Architecture

See also

"Oracle Database Instance"

Multitenant Architecture

The **multitenant architecture** enables an Oracle database to be a multitenant container database (CDB).
A non-CDB is a traditional Oracle database that cannot contain PDBs. A CDB is a single physical database that contains zero, one, or many user-created pluggable databases.
A pluggable database (PDB) is a portable collection of schemas, schema objects, and nonschema objects that appears to an Oracle Net client as a non-CDB.

Note

CDBs and non-CDBs have architectural differences. This manual assumes the architecture of a non-CDB unless otherwise indicated.

Benefits of the Multitenant Architecture

The multitenant architecture solves a number of problems posed by the traditional non-CDB architecture.

Large enterprises may use hundreds or thousands of databases, often running on different platforms on multiple physical servers. Modern servers are able to handle heavier workloads than before. A database may use only a fraction of the server hardware capacity. This approach wastes both hardware and human resources.

By consolidating multiple physical databases on separate computers into a single database on a single computer, the multitenant architecture provides the following benefits:

- Cost reduction for hardware
- Easier and more rapid movement of data and code
- Easier management and monitoring of the physical database
- Separation of data and code
- Separation of duties between a PDB administrator, who manages only the PDBs to which she or he is granted privileges, and the CDB administrator, who manages the entire CDB

Benefits for manageability include:

- Easier upgrade of data and code by unplugging and plugging in PDBs
- Easier testing by using PDBs for development before plugging them in to the production CDB
- Ability to flash back an individual PDB to a previous SCN
- Ability to set performance limits for memory and I/O at the PDB level
- Ability to install, upgrade, and manage a master application definition within an application container, which is a set of PDBs plugged in to a common application root