Chapter 4
Database Administration

The following sections provide information about the tools that can be used to maintain your Oracle Utilities Work and Asset Management Database.

Using SQL

SQL is a simple yet powerful database access language that is the standard language for relational database management systems. The SQL implemented by Oracle Utilities Work and Asset Management for Oracle is 100 percent compliant with the ANSI/ISO standard SQL data language.

SQL Statements

All operations on the information in an Oracle database are performed using SQL statements. A SQL statement is a string of SQL text that is given to Oracle to execute. A statement must be the equivalent of a complete SQL sentence, as in:

SELECT ename, deptno FROM emp;

Only a complete SQL statement can be executed, whereas a sentence fragment, such as the following, generates an error indicating that more text is required before a SQL statement can execute:

SELECT ename

A SQL statement can be thought of as a very simple, but powerful, computer program or instruction. SQL statements are divided into the following categories:

- Data Definition Language (DDL) statements
- Data Manipulation Language (DML) statements
- Transaction control statements
- Session control statements
- System control statements
- Embedded SQL statements

Data Definition Statements (DDL)

DDL statements define, maintain, and drop schema objects when they are no longer needed. DDL statements also include statements that permit a user to grant other users the privileges, or rights, to access the database and specific objects within the database.

DDL statements implicitly commit the preceding and start a new transaction.

Some examples of DDL statements follow:

CREATE TABLE plants

(COMMON_NAME VARCHAR2 (15), LATIN_NAME VARCHAR2 (40));

DROP TABLE plants;

GRANT SELECT ON emp TO scott;

REVOKE DELETE ON emp FROM scott;

Data Manipulation Statements (DML)

DML statements manipulate the database's data. For example, querying, inserting, updating, and deleting rows of a table are all DML operations; locking a table or view and examining the execution plan of an SQL statement are also DML operations.

DML statements are the most frequently used SQL statements. Some examples of DML statements follow:

SELECT ename, mgr, comm + sal FROM emp;

INSERT INTO emp VALUES

(1234, 'DAVIS', 'SALESMAN', 7698, '14-FEB-1988', 1600, 500, 30);

DELETE FROM emp WHERE ename IN ('WARD','JONES');

Transaction Control Statements

Transaction control statements manage the changes made by DML statements. They allow the user or application developer to group changes into logical transactions. Examples include COMMIT and ROLLBACK.

Session Control Statements

Session control statements allow a user to control the properties of his current session, including enabling and disabling roles and changing language settings. The two session control statements are ALTER SESSION and SET ROLE.

System Control Statements

System control commands change the properties of the Oracle server instance. The only system control command is ALTER SYSTEM; it allows you to change such settings as the minimum number of shared servers, to kill a session, and to perform other tasks.

Embedded SQL Statements

Embedded SQL statements incorporate DDL, DML, and transaction control statements in a procedural language program (such as those used with the Oracle Precompilers). Examples include OPEN, CLOSE, FETCH, and EXECUTE.

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.

Although most Oracle users rely on the automatic cursor handling of the Oracle utilities, the programmatic interfaces offer application designers more control over cursors. In application development, a cursor is a named resource available to a program and can be used specifically for the parsing of SQL statements embedded within the application.

Each user session can open multiple cursors up to the limit set by the initialization parameter OPEN_CURSORS. However, applications should close unneeded cursors to conserve system memory. If a cursor cannot be opened due to a limit on the number of cursors, the database administrator can alter the OPEN_CURSORS initialization parameter.

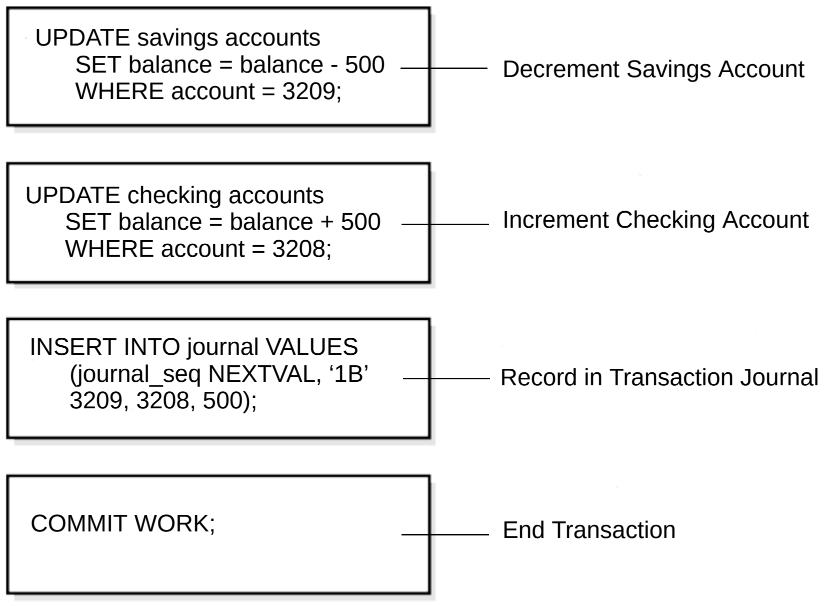
Some statements (primarily DDL statements) require Oracle to implicitly issue recursive SQL statements, which also require recursive cursors. For example, a CREATE TABLE statement causes many updates to various data dictionary tables to record the new table and columns. Recursive calls are made for those recursive cursors; one cursor may execute several recursive calls.

Transactions

A transaction is a logical unit of work that comprises one or more SQL statements executed by a single user. According to the ANSI/ISO SQL standard, with which Oracle is compatible, a transaction begins with the user's first executable SQL statement. A transaction ends when it is explicitly committed or rolled back (both terms are discussed later in this section) by that user.

Consider a banking database. When a bank customer transfers money from a savings account to a checking account, the transaction might consist of three separate operations: decrease the savings account, increase the checking account, and record the transaction in the transaction journal.

Oracle must guarantee that all three SQL statements are performed to maintain the accounts in proper balance. When something prevents one of the statements in the transaction from executing (such as a hardware failure), the other statements of the transaction must be undone; this is called "rolling back." If an error occurs in making either of the updates, then neither update is made.


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

```
UPDATE savings accounts
    SET balance = balance - 500
    WHERE account = 3209;

UPDATE checking accounts
    SET balance = balance + 500
    WHERE account = 3208;

INSERT INTO journal VALUES
    (journal_seq NEXTVAL, ‘1B’
    3209, 3208, 500);

COMMIT WORK;
```

Committing and Rolling Back Transactions

The changes made by the SQL statements that constitute a transaction can be either committed or rolled back. After a transaction is committed or rolled back, the next transaction begins with the next SQL statement.

Committing a transaction makes permanent the changes resulting from all SQL statements in the transaction. The changes made by the SQL statements of a transaction become visible to other user sessions’ transactions that start only after the transaction is committed.

Rolling back a transaction retracts any of the changes resulting from the SQL statements in the transaction. After a transaction is rolled back, the affected data is left unchanged as if the SQL statements in the transaction were never executed.

PL/SQL

PL/SQL is Oracle’s procedural language extension to SQL. PL/SQL combines the ease and flexibility of SQL with the procedural functionality of a structured programming language, such as IF ... THEN, WHILE, and LOOP.

When designing a database application, a developer should consider the advantages of using stored PL/SQL:

- Because PL/SQL code can be stored centrally in a database, network traffic between applications and the database is reduced, so application and system performance increases.
- Data access can be controlled by stored PL/SQL code. In this case, the users of PL/SQL can access data only as intended by the application developer (unless another access route is granted).
- PL/SQL blocks can be sent by an application to a database, executing complex operations without excessive network traffic.

Even when PL/SQL is not stored in the database, applications can send blocks of PL/SQL to the database rather than individual SQL statements, thereby again reducing network traffic.

The following sections describe the different program units that can be defined and stored centrally in a database.

Procedures and Functions

Procedures and functions consist of a set of SQL and PL/SQL statements that are grouped together as a unit to solve a specific problem or perform a set of related tasks. A procedure is created and stored in compiled form in the database and can be executed by a user or a database application.

Procedures and functions are identical except that functions always return a single value to the caller, while procedures do not return values to the caller.

Packages

Packages provide a method of encapsulating and storing related procedures, functions, variables, and other package constructs together as a unit in the database. While packages allow the administrator or application developer the ability to organize such routines, they also offer increased functionality (for example, global package variables can be declared and used by any procedure in the package) and performance (for example, all objects of the package are parsed, compiled, and loaded into memory once).

Keys

The term "key" is used in the definitions of several types of integrity constraints. A key is the column or set of columns included in the definition of certain types of integrity constraints. Keys describe the relationships between the different tables and columns of a relational database. The different types of keys include:

- **primary key** - The column or set of columns included in the definition of a table's PRIMARY KEY constraint. A primary key's values uniquely identify the rows in a table. Only one primary key may be defined per table.
- **unique key** - The column or set of columns included in the definition of a UNIQUE constraint.
- **foreign key** - The column or set of columns included in the definition of a referential integrity constraint.
- **referenced key** - The unique key or primary key of the same or different table that is referenced by a foreign key.

Individual values in a key are called key values.

Database Triggers

Oracle allows you to write procedures that are automatically executed as a result of an insert in, update to, or delete from a table. These procedures are called database triggers.

Database triggers can be used in a variety of ways for the information management of your database. For example, they can be used to automate data generation, audit data modifications, enforce complex integrity constraints, and customize complex security authorizations.

Centralized actions can be defined using a non-declarative approach (writing PL/SQL code) with database triggers. A database trigger is a stored procedure that is fired (implicitly executed) when an INSERT, UPDATE, or DELETE statement is issued against the associated table. Database triggers can be used to customize a database management system with such features as value-based auditing and the enforcement of complex security checks and integrity rules. For example, a database trigger might be created to allow a table to be modified only during normal business hours.

Custom SQL Search Query Examples

Note: System Administrators can disable users' ability to use Custom SQL by adding the DISABLE CUSTOM SQL Responsibility to the user's profile.

1. **Active or Pending Approval requisitions. Selecting stock reorder requisitions in created or pending approval status. FMC leaves requisitions in created to accumulate more parts and reduce the number of PO's. This way they can review and combine with any requisitions input manually or created by batch after status was changed to PA.**

```
PLANT||REQUISITION_NO IN
(SELECT PLANT||REQUISITION_NO
FROM SA_REQUISITION
WHERE
REQUISITION_STATUS IN ('CREATED','PENDING APPROVAL')
AND REQUISITION_TYPE LIKE 'S%'
AND NVL(NEXT_APPROVER,'STORES') LIKE 'STORES%' )
[order by vendor code]
```

2. **Obsolete stock codes. Searching inventory stock codes in catalog where storeroom is active and stores personnel have marked it as obsolete or same as another part (in description). These parts are reviewed to see if FMC need to report value for write off to accounting.**

```
PLANT||STOCK_CODE||STOREROOM IN
(SELECT S.PLANT||S.STOCK_CODE||S.STOREROOM
 FROM SA_STOREROOM S, SA_CATALOG C
 WHERE S.STOREROOM_STATUS = 'ACTIVE'
 AND (UPPER(C.STOCK_DESC) LIKE '%SAME AS%'
        OR UPPER(C.STOCK_DESC) LIKE '%OBSOLETE%')
 AND C.STOCK_TYPE = 'INVENTORY'
 AND C.STOCK_CODE = S.STOCK_CODE)
```

3. **Vendors with null manufacturer codes. Selecting catalog stock codes by vendor (edit selection and save) for parts with no manufacturer code.**

```
PLANT||STOCK_CODE||STOREROOM IN
(SELECT S.PLANT||S.STOCK_CODE||S.STOREROOM
 FROM SA_STOREROOM S, SA_CATALOG_MFR_VENDOR V
 WHERE V.MANUFACTURER_CODE IS NULL
 AND V.VENDOR_CODE LIKE '4924%'
 AND V.PRIMARY_VENDOR_IND = 'Y'
 AND S.STOCK_CODE = V.STOCK_CODE)
```

4. **Line item description in PO’s. Searching for purchase orders by line item description, where description contains the word ‘MOTOR’.**
   ```
   PLANT||PO_NO IN
   (SELECT I.PLANT||I.PO_NO
    FROM SA_PURCHASE_ORDER_ITEM I
    WHERE UPPER(I.ITEM_DESC) LIKE ‘%MOTOR%’)
   ```
5. **Detail views. Searching for work orders where any task downtime type is ‘Shutdown’.**
   ```
   PLANT||WORK_ORDER_NO IN
   (SELECT TASK.PLANT||TASK.WORK_ORDER_NO
   FROM SA_WORK_ORDER_TASK TASK
   WHERE TASK.DOWNTIME_TYPE = ‘SHUTDOWN’)
   ```
6. **Comparison. Searching for stock codes where the inventory quantity has fallen below minimum quantity.**
   ```
   PLANT||STOCK_CODE||STOREROOM IN
   ( SELECT S.PLANT||S.STOCK_CODE||S.STOREROOM
    FROM SA_STOREROOM S
    WHERE NVL(S.INVENTORY_QUANTITY,0) < NVL(MINIMUM_QUANTITY,0))
   ```
7. **Date Ranges & Custom Order by. Search for work orders due during Feb 1, 1999 and Feb 20, 2000 in ‘ACTIVE’ or ‘PENDING APPROVAL’ work status. Results of search is ordered by Required Date.**
   ```
   WORK_REQUIRED_DATE >= TO_DATE(’01-FEB-1999',’DD-MON-YYYY’)
   AND WORK_REQUIRED_DATE <= TO_DATE(‘20-FEB-2000’, ’DD-MON-YYYY’)
   AND WORK_STATUS IN (‘PENDING APPROVAL’, ‘ACTIVE’)
   [CUSTOM ORDER BY ‘WORK_REQUIRED_DATE’]
   ```
8. **NULL operator. Search for Checkout Request records that are not associated with a Work Order record.**
   ```
   WORK_ORDER_NO IS NULL
   ```
9. **Search in the Accounting Log for the vendor with largest transaction cost.**
   ```
   PLANT||ACCOUNT_NO||EXPENSE_CODE||
   PERIOD_YEAR||PERIOD_MONTH||TRANSACTION_DATE||
   SEQUENCE_NUMBER IN
   (SELECT L.PLANT||L.ACCOUNT_NO||L.EXPENSE_CODE||
   L.PERIOD_YEAR||L.PERIOD_MONTH||L.TRANSACTION_DATE||
   L.SEQUENCE_NUMBER
   FROM SA_ACCOUNT_LOG L
   WHERE TRANSACTION_AMOUNT = (SELECT MAX(TRANSACTION_
   AMOUNT)
   FROM SA_ACCOUNT_LOG
   WHERE VENDOR_CODE IS NOT NULL))
   ```
10. **Reorder reviews in created status that are on BPO’s**
    ```
    PLANT||STOCK_CODE IN
    ```

```
(SELECT PLANT||STOCK_CODE
FROM SA_BLANKET_CONTRACT_ITEM
WHERE BLANKET_CONTRACT_NO IN
(SELECT BLANKET_CONTRACT_NO FROM SA_BLANKET_CONTRACT
WHERE BLANKET_STATUS='ACTIVE'))
AND REORDER_STATUS='CREATED'
```

11. **Reorder reviews in 'Created' status that are not on BPO's**

```
PLANT||STOCK_CODE IN
(SELECT PLANT||STOCK_CODE
FROM SA_STOREROOM_REORDER_REVIEW
MINUS
SELECT PLANT||STOCK_CODE
FROM SA_BLANKET_CONTRACT_ITEM
WHERE BLANKET_CONTRACT_NO IN
(SELECT BLANKET_CONTRACT_NO FROM SA_BLANKET_CONTRACT
WHERE BLANKET_STATUS='ACTIVE'
AND BLANKET_ITEM_STATUS='ACTIVE'
AND REORDER_STATUS='CREATED'))
```

12. **Storeroom records with quantity > 0 but total cost = 0.**

```
PLANT||STOCK_CODE IN
(SELECT PLANT||STOCK_CODE
FROM SA_STOREROOM
WHERE NVL(INVENTORY_QUANTITY,0) !=0
AND NVL(TOTAL_VALUE,0) = 0)
```

13. **Storeroom records with available quantity < 0**

```
PLANT||STOCK_CODE IN
(SELECT PLANT||STOCK_CODE FROM SA_STOREROOM
WHERE NVL(INVENTORY_QUANTITY,0) != 0
AND NVL(TOTAL_VALUE,0) = 0)
```

14. **Work Order tasks that are charged to construction work orders.**

```
PLANT||WORK_ORDER_NO IN
(SELECT PLANT||WORK_ORDER_NO
FROM SA_WORK_ORDER
WHERE SUBSTR(ATTRIBUTE1,1,2)='A0')
AND CREATION_DATE >= TRUNC(SYSDATE) – 3
AND TASK_STATUS = 'ACTIVE'
```

15. **PO's received in the system but not received in AF/S (external) system.**

```
PLANT||PO_NO IN
(SELECT DISTINCT PLANT||PO_NO FROM SA_PURCHASE_ORDER_ITEM
```

WHERE NVL(RECEIVED_NET_QUANTITY,0) != NVL(ATTRIBUTE4,0)

AND PO_ITEM_STATUS != 'CANCELED'

AND TRUNC(LAST_RECEIVED_DATE) != TRUNC(SYSDATE)

AND PO_STATUS != 'CANCELED'

AND PO_REVISION_NO = NVL(ATTRIBUTE1,''))

16. **PO's not fully received by Promise Date**
PO_STATUS = 'ISSUED'

AND ((REQUIRED_DATE IS NULL)

OR (TRUNC(REQUIRED_DATE) <= TRUNC(SYSDATE)))

17. **PO's with total cost > $250,000.00**
TOTAL_AMOUNT > 250000

18. **UDF's. Invoices posted but no check date from AF/S**
INVOICE_STATUS = 'POSTED'

AND ATTRIBUTE5 IS NULL

19. **Reorder review – lost items**
PLANT||STOCK_CODE IN

(SELECT PLANT||STOCK_CODE

FROM SA_STOREROOM

WHERE (INVENTORY_QUANTITY + ON_ORDER_QUANTITY)

< MINIMUM_QUANTITY

AND TRUNC(LAST_UPDATE_DATE) < TRUNC(SYSDATE)

MINUS

SELECT PLANT||STOCK_CODE

FROM SA_STOREROOM_REORDER_REVIEW

WHERE (REORDER_STATUS = 'CREATED'

OR REORDER_STATUS = 'APPROVED')

MINUS

SELECT PLANT||STOCK_CODE

FROM SA_PURCHASE_ORDER_ITEM

WHERE PO_ITEM_STATUS != 'CANCELED'

AND PO_NO IN

(SELECT PO_NO FROM SA_PURCHASE_ORDER

WHERE (PO_STATUS = 'CREATED'

OR PO_STATUS = 'APPROVED')))

20. **Reorder review created records > $99.00**
PLANT||STOCK_CODE IN

(SELECT PLANT||STOCK_CODE

FROM SA_STOREROOM

WHERE AVERAGE_UNIT_PRICE > 99.00)

AND REORDER_STATUS = 'CREATED'

21. **Searching for work orders where costs in the 'Cost Summary' detail option are greater than $1000.**
    PLANT||WORK_ORDER_NO IN

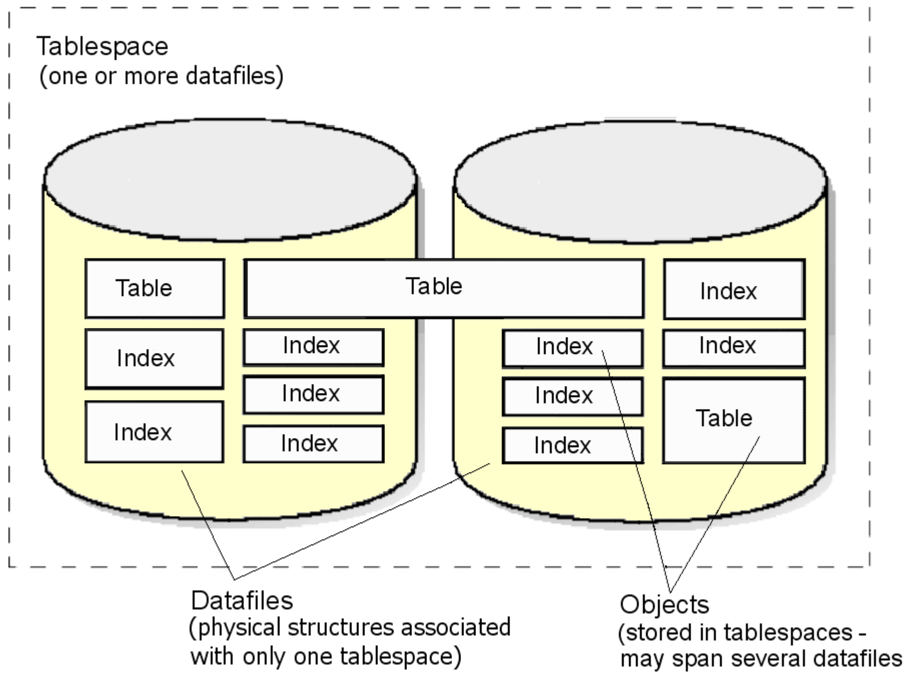
    (SELECT PLANT||WORK_ORDER_NO

    FROM SV_WORK_ORDER_COST

    WHERE ACTUAL_AMOUNT > '1000')

Databases, Tablespaces & Datafiles

Oracle stores data logically in tablespaces and physically in datafiles associated with the corresponding tablespace. The figure below illustrates this relationship.



Note: A database consists of a number of tablespaces. Each tablespace contains one or more datafiles.

Databases, tablespaces, and datafiles are closely related, but they have important differences:

Databases and Tablespaces - An Oracle database consists of one or more logical storage units called tablespaces, which collectively store all of the database's data.

Tablespaces and Datafiles - Each tablespace in an Oracle database consists of one or more files called datafiles, which are physical structures that conform with the operating system in which Oracle is running.

Databases and Datafiles - A database's data is collectively stored in the datafiles that constitute each tablespace of the database. For example, the simplest Oracle database would have one tablespace and one datafile. Another database might have three tablespaces, each consisting of two datafiles (for a total of six datafiles).

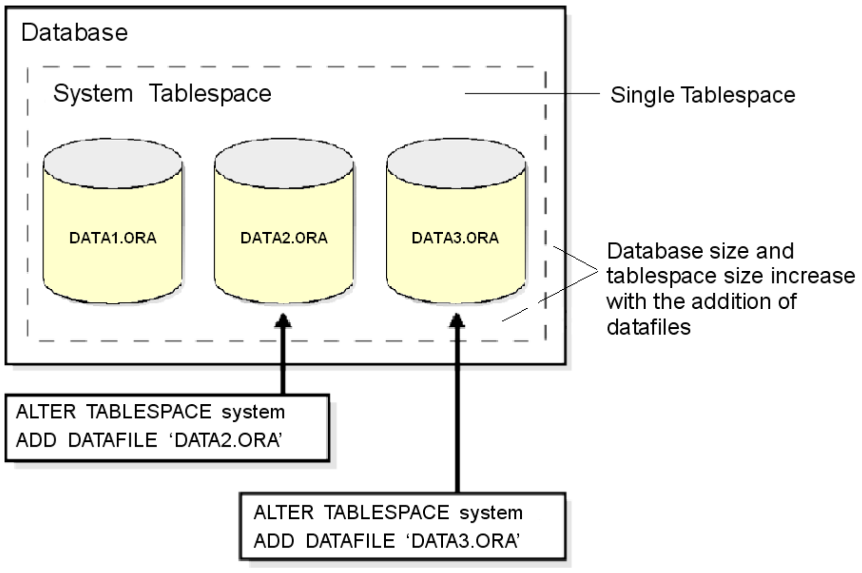
Allocating More Space for a Database

You can enlarge a database in three ways:

1. **Add a datafile to a tablespace.**
2. **Add a new tablespace.**
3. **Increase the size of a datafile.**

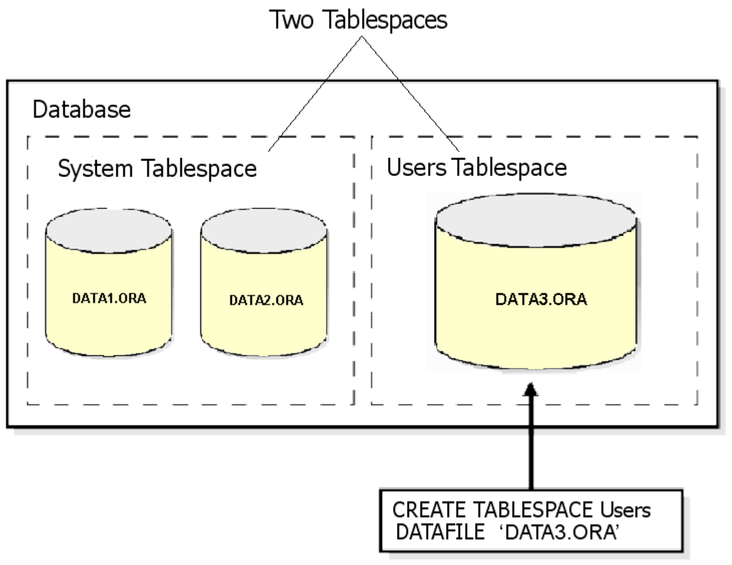
Adding a Datafile to a Tablespace

When you add another datafile to an existing tablespace, you increase the amount of disk space allocated for the corresponding tablespace. The figure below illustrates this kind of space increase.


Database
System Tablespace
Single Tablespace
DATA1.ORA
DATA2.ORA
DATA3.ORA
Database size and tablespace size increase with the addition of datafiles
ALTER TABLESPACE system ADD DATAFILE 'DATA2.ORA'
ALTER TABLESPACE system ADD DATAFILE 'DATA3.ORA'

Adding a New Tablespace

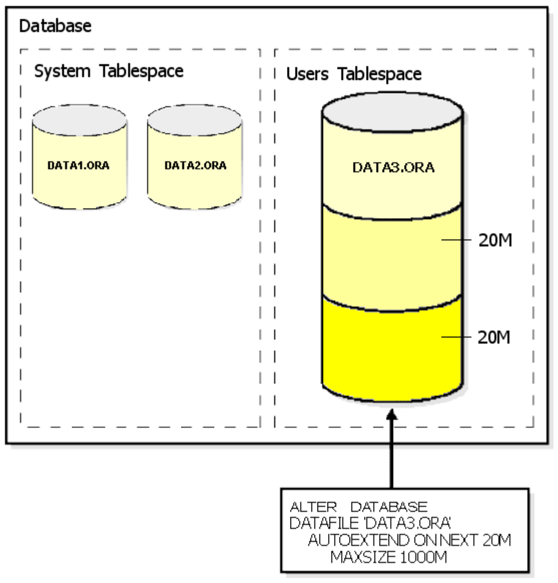
Alternatively, you can create a new tablespace containing at least one additional datafile to increase the size of a database as illustrated below.



Note: The size of a tablespace is the size of the datafiles that makes up the tablespace. The size of a database is the collective size of the tablespaces that make up the database.

Increasing the Size of a Datafile

The third option for enlarging a database is to change a datafile's size or to allow datafiles in existing tablespaces to grow dynamically as more space is needed. Do this by altering existing files or by adding files with dynamic extension properties as illustrated below.



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.

This section includes the following topics about tablespaces:

- The SYSTEM Tablespace
- Multiple Tablespaces
- Space Management in Tablespaces
- Online and Offline Tablespaces
- Transporting Tablespaces between Databases

The SYSTEM Tablespace

Every Oracle database contains a tablespace named SYSTEM, which Oracle creates automatically when the database is created.

The SYSTEM tablespace always contains the data dictionary tables for the entire database. The data dictionary tables are stored in Datafile 1.

Multiple Tablespaces

A small database might need only the SYSTEM tablespace; however, Oracle Corporation recommends that you create at least one additional tablespace to store user data separate from data dictionary information. This gives you more flexibility in various database administration operations and reduces contention among dictionary objects and schema objects for the same datafiles.

You can use multiple tablespaces to:

- control disk space allocation for database data
- assign specific space quotas for database users
- control availability of data by taking individual tablespaces online or offline
- perform partial database backup or recovery operations

- allocate data storage across devices to improve performance

A database administrator (DBA) can create new tablespaces, add datafiles to tablespaces, set and alter default segment storage settings for segments created in a tablespace, make a tablespace read-only or read-write, make a tablespace temporary or permanent, and drop tablespaces.

Space Management in Tablespaces

An extent is a logical unit of storage space made up of a number of contiguous data blocks.

Tablespaces allocate space in extents. Tablespaces can use two different methods to keep track of their free and used space:

- extent management by the data dictionary (dictionary-managed tablespaces)
- extent management by the tablespace (locally-managed tablespaces)

When you create a tablespace, you choose one of these methods of space management. You cannot alter the method at a later time.

Dictionary-Managed Tablespaces

For a tablespace that uses the data dictionary to manage its extents, Oracle updates the appropriate tables in the data dictionary whenever an extent is allocated or freed for reuse. Oracle also stores rollback information about each update of the dictionary tables. Since dictionary tables and rollback segments are part of the database, the space that they occupy is subject to the same space management operations as all other data.

This is the default method of space management in a tablespace. It was the only method available in Oracle releases 8.0 and earlier.

Locally-Managed Tablespaces

Local management of extents automatically tracks adjacent free space, eliminating the need to coalesce free extents.

A tablespace that manages its own extents maintains a bitmap in each datafile to keep track of the free or used status of blocks in that datafile. Each bit in the bitmap corresponds to a block or a group of blocks. When an extent is allocated or freed for reuse, Oracle changes the bitmap values to show the new status of the blocks. These changes do not generate rollback information because they do not update tables in the data dictionary (except for special cases such as tablespace quota information).

One of the greatest advantages of locally-managed tablespaces over dictionary-managed tablespaces is that local management of extents avoids recursive space management operations which can occur in dictionary-managed tablespaces if consuming or releasing space in an extent results in another operation that consumes or releases space in a rollback segment or data dictionary table. Local management of extents automatically tracks adjacent free space, eliminating the need to coalesce free extents.

The sizes of extents that are managed locally can be determined automatically by the system. Alternatively, all extents can have the same size in a locally-managed tablespace.

The LOCAL option of the EXTENT MANAGEMENT clause specifies this method of space management in various CREATE commands:

- For the SYSTEM tablespace, you can specify EXTENT MANGEMENT LOCAL in the CREATE DATABASE command. If the SYSTEM tablespace is locally managed, other tablespaces in the database can be dictionary-managed but you must create all rollback segments in locally-managed tablespaces.
- For a permanent tablespace other than SYSTEM, you can specify EXTENT MANGEMENT LOCAL in the CREATE TABLESPACE command.
- For a temporary tablespace, you can specify EXTENT MANGEMENT LOCAL in the CREATE TEMPORARY TABLESPACE command.

Online and Offline Tablespaces

A database administrator can bring any tablespace other than the SYSTEM tablespace online (accessible) or offline (not accessible) whenever the database is open. The SYSTEM tablespace is always online when the database is open because the data dictionary must always be available to Oracle.

A tablespace is normally online so that the data contained within it is available to database users. However, the database administrator might take a tablespace offline

- To make a portion of the database unavailable, while allowing normal access to the remainder of the database.
- To perform an offline tablespace backup (although a tablespace can be backed up while online and in use).
- To make an application and its group of tables temporarily unavailable while updating or maintaining the application.

You cannot take a tablespace offline if it contains any rollback segments that are in use.

When a Tablespace Goes Offline

When a tablespace goes offline, Oracle does not permit any subsequent SQL statements to reference objects contained in that tablespace. Active transactions with completed statements that refer to data in that tablespace are not affected at the transaction level. Oracle saves rollback data corresponding to those completed statements in a deferred rollback segment (in the SYSTEM tablespace). When the tablespace is brought back online, Oracle applies the rollback data to the tablespace, if needed.

When a tablespace goes offline or comes back online, this is recorded in the data dictionary in the SYSTEM tablespace. If a tablespace was offline when you shut down a database, the tablespace remains offline when the database is subsequently mounted and reopened.

You can bring a tablespace online only in the database in which it was created because the necessary data dictionary information is maintained in the SYSTEM tablespace of that database. An offline tablespace cannot be read or edited by any utility other than Oracle. Thus, offline tablespaces cannot be transferred from database to database.

Tablespaces are automatically switched offline by Oracle when certain errors occur.

Oracle automatically switches a tablespace from online to offline when certain errors are encountered (for example, when the database writer process, DBWn, fails in several attempts to write to a datafile of the tablespace). Users trying to access tables in the offline tablespace receive an error. If the problem that causes this disk I/O to fail is media failure, you must recover the tablespace after you correct the hardware problem.

Using Tablespaces for Special Procedures

If you create multiple tablespaces to separate different types of data, you take specific tablespaces offline for various procedures; other tablespaces remain online and the information in them is still available for use. However, special circumstances can occur when tablespaces are taken offline. For example, if two tablespaces are used to separate table data from index data, the following is true:

- If the tablespace containing the indexes is offline, queries can still access table data because queries do not require an index to access the table data.
- If the tablespace containing the tables is offline, the table data in the database is not accessible because the tables are required to access the data.

In summary, if Oracle has enough information in the online tablespaces to execute a statement, it will do so. If it needs data in an offline tablespace, then it causes the statement to fail.

Transporting Tablespaces Between Databases

The transportable tablespace feature enables you to move a subset of an Oracle database from one Oracle database to another. You can clone a tablespace from one tablespace and plug it into another database, copying the tablespace between databases, or you can unplug a tablespace from one Oracle database and plug it into another Oracle database, moving the tablespace between databases.

Moving data by transporting tablespaces can be orders of magnitude faster than either export/import or unload/load of the same data, because transporting a tablespace involves only copying datafiles and integrating the tablespace metadata. When you transport tablespaces you can also move index data, so that you do not have to rebuild the indexes after importing or loading the table data.

You can only transport tablespaces between Oracle databases that use the same data block size and character set, and that run on compatible platforms from the same hardware vendor.

Moving or Copying a Tablespace to Another Database

To move or copy a set of tablespaces, you must make the tablespaces read-only, copy the datafiles of these tablespaces, and use export/import to move the database information (metadata) stored in data dictionary. Both the datafiles and the metadata export file must be copied to the target database. The transport of these files can be done using any facility for copying flat files, such as the operating system copying facility, FTP, or publishing on CDs.

After copying the datafiles and importing the metadata, you can optionally put the tablespaces in read-write mode.

Datafiles

A tablespace in an Oracle database consists of one or more physical datafiles. A datafile can be associated with only one tablespace and only one database.

Oracle creates a datafile for a tablespace by allocating the specified amount of disk space plus the overhead required for the file header. When a datafile is created, the operating system in which Oracle is running is responsible for clearing old information and authorizations from a file before allocating it to Oracle. If the file is large, this process might take a significant amount of time.

The first tablespace in any database is always the SYSTEM tablespace, so Oracle automatically allocates the first datafiles of any database for the SYSTEM tablespace during database creation.

Datafile Contents

When a datafile is first created, the allocated disk space is formatted but does not contain any user data; however, Oracle reserves the space to hold the data for future segments of the associated tablespace—it is used exclusively by Oracle. As the data grows in a tablespace, Oracle uses the free space in the associated datafiles to allocate extents for the segment.

The data associated with schema objects in a tablespace is physically stored in one or more of the datafiles that constitute the tablespace. Note that a schema object does not correspond to a specific datafile; rather, a datafile is a repository for the data of any schema object within a specific tablespace. Oracle allocates space for the data associated with a schema object in one or more datafiles of a tablespace. Therefore, a schema object can “span” one or more datafiles. Unless table “striping” is used (where data is spread across more than one disk), the database administrator and end users cannot control which datafile stores a schema object.

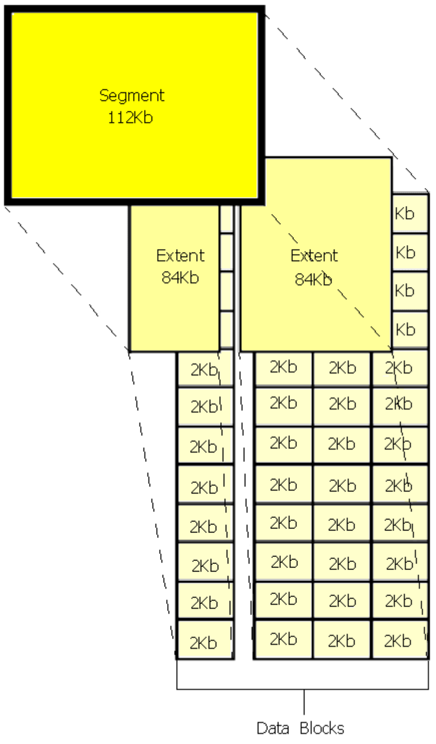
Size of Datafiles

You can alter the size of a datafile after its creation or you can specify that a datafile should dynamically grow as schema objects in the tablespace grow. This functionality enables you to have fewer datafiles per tablespace and can simplify administration of datafiles.

Data Blocks, Extents, & Segments

The units of database space allocation are data blocks, extents, and segments.

Oracle allocates logical database space for all data in a database. The units of database space allocation are data blocks, extents, and segments. The following illustration shows the relationships among these data structures:



Data Blocks

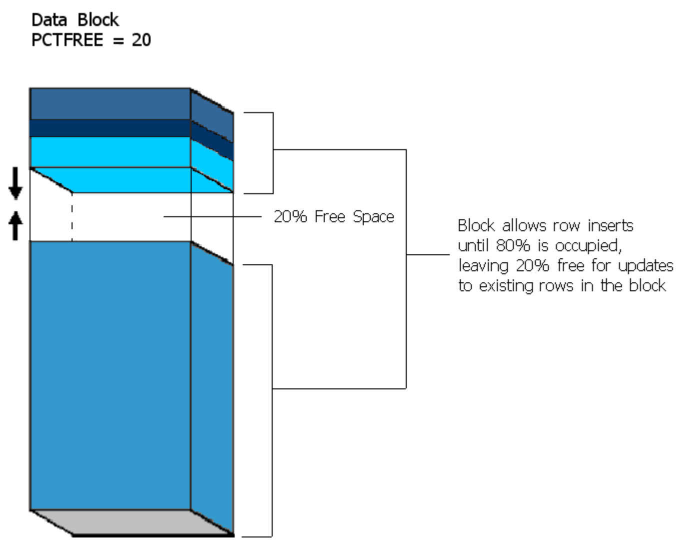
At the finest level of granularity, Oracle stores data in data blocks (also called logical blocks, Oracle blocks, or pages). One data block corresponds to a specific number of bytes of physical database space on disk.

The PCTFREE Parameter

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. For example, assume that you specify the following parameter within a CREATE TABLE statement:

PCTFREE 20

This states that 20% of each data block in this table's data segment will be kept free and available for possible updates to the existing rows already within each block. New rows can be added to the row data area, and corresponding information can be added to the variable portions of the overhead area, until the row data and overhead total 80% of the total block size.

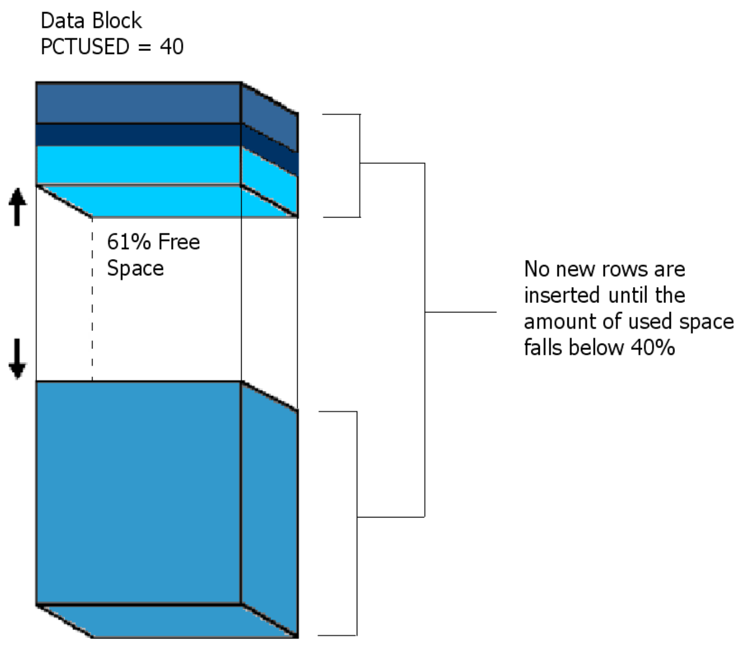

Data Block
PCTFREE = 20
20% Free Space
Block allows row inserts until 80% is occupied, leaving 20% free for updates to existing rows in the block

The PCTUSED Parameter

The PCTUSED parameter sets the minimum percentage of a block that can be used for row data plus overhead before new rows will be added to the block. After a data block is filled to the limit determined by PCTFREE, Oracle considers the block unavailable for the insertion of new rows until the percentage of that block falls below the parameter PCTUSED. Until this value is achieved, Oracle uses the free space of the data block only for updates to rows already contained in the data block. For example, assume that you specify the following parameter in a CREATE TABLE statement:

PCTUSED 40

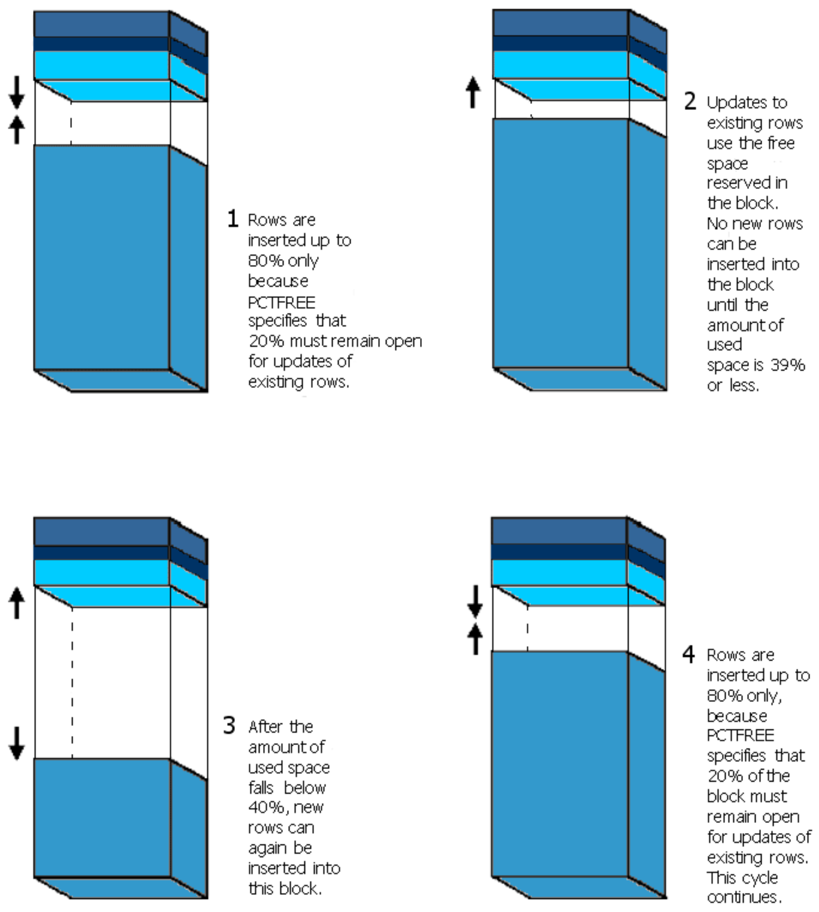
In this case, a data block used for this table's data segment is considered unavailable for the insertion of any new rows until the amount of used space in the block falls to 39% or less (assuming that the block's used space has previously reached PCTFREE). Figure 3 illustrates this.


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

This figure illustrates the interaction between PCTFREE and PCTUSED.

How PCTFREE and PCTUSED Work Together

PCTFREE and PCTUSED work together to optimize the utilization of space in the data blocks of the extents within a data segment.


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

In a newly allocated data block, the space available for inserts is the block size minus the sum of the block overhead and free space (PCTFREE). Updates to existing data can use any available space in the block; therefore, updates can reduce the available space of a block to less than PCTFREE, the space reserved for updates but not accessible to inserts.

For each data and index segment, Oracle maintains one or more free lists—lists of data blocks that have been allocated for that segment’s extents and have free space greater than PCTFREE; these blocks are available for inserts. When you issue an INSERT statement, Oracle checks a free list of the table for the first available data block and uses it if possible. If the free space in that block is not large enough to accommodate the INSERT statement, and the block is at least PCTUSED, Oracle takes the block off the free list. Multiple free lists per segment can reduce contention for free lists when concurrent inserts take place.

After you issue a DELETE or UPDATE statement, Oracle processes the statement and checks to see if the space being used in the block is now less than PCTUSED. If it is, the block goes to the beginning of the transaction free list, and it is the first of the available blocks to be used in that transaction. When the transaction commits, free space in the block becomes available for other transactions.

Extents

The next level of logical database space is an extent. An extent is a logical unit of database storage space allocation made up of a number of contiguous data blocks. One or more extents in turn make up a segment. When the existing space in a segment is completely used, Oracle allocates a new extent for the segment.

When Extents Are Allocated

When you create a table, Oracle allocates to the table's data segment an initial extent of a specified number of data blocks. Although no rows have been inserted yet, the Oracle data blocks that correspond to the initial extent are reserved for that table's rows.

If the data blocks of a segment's initial extent become full and more space is required to hold new data, Oracle automatically allocates an incremental extent for that segment. An incremental extent is a subsequent extent of the same or greater size than the previously allocated extent in that segment. (The next section explains the factors controlling the size of incremental extents.)

For maintenance purposes, the header block of each segment contains a directory of the extents in that segment.

Rollback segments always have at least two extents.

Segments

The level of logical database storage above an extent is called a segment. 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. For example, each table's data is stored in its own data segment, while each index's data is stored in its own index segment. If the table or index is partitioned, each partition is stored in its own segment.

Oracle allocates space for segments in units of one extent. When the existing extents of a segment are full, Oracle allocates another extent for that segment. Since extents are allocated as needed, the extents of a segment may or may not be contiguous on disk.

A segment and all its extents are stored in one tablespace. Within a tablespace, a segment can include extents from more than one file, that is, the segment can span datafiles. However, each extent can contain data from only one datafile.

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. An Oracle index provides a faster access path to table data. Indexes are the primary means of reducing disk I/O when properly used.

You can create an unlimited number of indexes for a table provided that the combination of columns differs for each index. You can create more than one index using the same columns provided that 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);
```

You cannot create an index that references only one column in a table if another such index already exists.

The absence or presence of an index does not require a change in the wording of any SQL statement. An index is merely a fast access path to the data; it affects only the speed of execution. Given a data value that has been indexed, the index points directly to the location of the rows containing that value.

Indexes are logically and physically independent of the data in the associated table. You can create or drop an index at anytime without affecting the base tables or other indexes. If you drop an index, all applications continue to work; however, access of previously indexed data might be slower. Indexes, as independent structures, require storage space.

Oracle automatically maintains and uses indexes once they are created, and automatically reflects changes to data, such as adding new rows, updating rows, or deleting rows, in all relevant indexes with no additional action by users.

Retrieval performance of indexed data remains almost constant, even as new rows are inserted. However, the presence of many indexes on a table decreases the performance of updates, deletes, and inserts because Oracle must also update the indexes associated with the table.

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 columns that define the index. Nonunique indexes do not impose this restriction on the column values.

Oracle recommends that you do not explicitly define unique indexes on tables; uniqueness is strictly a logical concept and should be associated with the definition of a table. Alternatively, 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 (also called a concatenated 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.

No more than 32 columns can form a regular composite index.

Indexes and Keys

Although the terms are often used interchangeably, you should understand the 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.

Since Oracle uses indexes to enforce some integrity constraints, the terms key and index are often are used interchangeably; however, they should not be confused with each other.

How Indexes Are Stored

When you create an index, Oracle automatically allocates an index segment to hold the index’s data in a tablespace. You control allocation of space for an index’s segment and use of this reserved space in the following ways:

- Set the storage parameters for the index segment to control the allocation of the index segment’s extents.
- Set the PCTFREE parameter for the index segment to control the free space in the data blocks that constitute the index segment’s extents.

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 do not have to place an index in the same tablespace as its associated table. Furthermore, 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 because Oracle can retrieve both index and table data in parallel.

Tables

Tables are relational database structures which are analogous to the entity of a data model. An index-organized table differs from an ordinary table in that the data for the table is held in its associated index. Changes to the table data, such as adding new rows, updating rows, or deleting rows, result only in updating the index.

Ordinary Table

Row ID uniquely identifies a row; primary key can be optionally specified

Physical Row ID in ROWID pseudocolumn allows building secondary indexes

Row ID based access

Sequential scan returns all rows

UNIQUE constraint and triggers allowed

Can be stored in a cluster with other tables

Can contain a column of the LONG datatype and columns of LOB datatypes

Distribution and replication supported

Index-Organized Table

Primary key uniquely identifies a row; primary key must be specified

Logical Row ID in ROWID pseudocolumn allows building secondary indexes

Primary key based access

Full-index scan returns all rows in primary key order

UNIQUE constraint not allowed but triggers are allowed

Cannot be stored in a cluster

Can contain LOB columns but not LONG columns

Distribution and replication not supported

Benefits of Index-Organized Tables

Since data rows are stored in the index, index-organized tables provide faster key-based access to table data for queries that involve exact match or range search, or both. The storage requirements are reduced because key columns are not duplicated as they are in an ordinary table and its index. The data row stored with the key in an index-organized table only contains non-key column values. Also, placing the data row with the key eliminates the additional storage that an index on an ordinary table requires for physical Row IDs, which link the key values to corresponding rows in the table.

DBA Glossary

Base Table - The underlying table in the data dictionary. Base tables store information about the associated database. Only Oracle should write to and read these tables. Users rarely access them directly because they are normalized, and most of the data is stored in a cryptic format.

Block - A set of related fields in a module. Blocks do not necessarily correspond directly with windows. While one block typically represents one screen (Header, one of the Views, etc.), this is not always the case. Some windows use two or more blocks. In other cases, a view will share the same block as the header. To check the name of a block associated with a portion of a window, place the cursor in a field and select About Item from the Help menu.

Code Table - Code tables are used to keep track of codes that stand for names, titles, labels and other information that is used frequently.

Composite Index - (also called a concatenated index) 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.

Cursor - A name for a private SQL area—an area in memory in which a parsed statement and other information for processing the statement are kept.

Custom Menu - A user defined menu. The menu items that users create can open reports or custom API calls that are developed using SQL. Once the fields in this form are completed, the system adds a menu item to the menu bar of the selected module. To users, the added menu items look like standard options, while adding greater accessibility and functionality specific to an organization.

Custom SQL - Manual commands that can be entered by a user to perform searches based on criteria that are not available on the standard search screens.

Customize - To modify according to individual specifications.

Database Trigger - A procedure that can be written to be executed automatically as a result of an insert in, update to, or delete from a table.

Datablock - The smallest unit of database space allocation. Oracle stores data in data blocks (also called logical blocks, Oracle blocks, or pages). One data block corresponds to a specific number of bytes of physical database space on disk. Several data blocks make up an extent.

Data Dictionary - A read-only set of tables that provides information about the database's associated database.

Datafile - Physical structures where data is stored in a database. Datafiles typically conform with the operating system in which Oracle is running.

DBA (Database Administrator) - The person who keeps the database running smoothly. Among other tasks, the DBA helps make sure codes are correct, handles problems and installs new features as they become available.

Extent - A specific number of contiguous data blocks allocated for storing a specific type of information. Several extents make up a segment.

Field - A space allocated for a particular item of information on a record. Fields are grouped into records and any given field may be used in several different kinds of records. For example a stock number may be used in a record for keeping track of inventory, and also be used in a record about the service life and reliability of parts.

Foreign Key - A field that identifies a specific record in a different module. The column or set of columns included in the definition of a referential integrity constraint.

Functions - A set of SQL and PL/SQL statements that are grouped together as a unit to solve a specific problem or perform a set of related tasks. Procedures and functions are identical except that functions always return a single value to the caller, while procedures do not return values to the caller.

Header - The main information window in a module.

Index - A structure actually stored in the database, which users create, alter, and drop using SQL statements.

Join - A SQL function that allows you to search by information located in a different table.

Key - A column or set of columns included in the definition of certain types of integrity constraints. Keys describe the relationships between the different tables and columns of a relational database. The different types of keys include: primary key, unique key, foreign key, and referenced key.

List of Values (LOV) - A predetermined list of possible values that can fill in a field. The lists serve to help you locate information that you might not remember. They also ensure that the information entered into the database is both consistent and correct.

Nonunique Index - An index where two or more rows of a table can have duplicate values in the columns that define the index.

ODBC (Open DataBase Connectivity) - ODBC is a programming interface language that allows database programs to communicate using a common set of SQL queries. Oracle Utilities Work and Asset Management can use ODBC to exchange information with MS Project, ArcView, and other programs. In order for you to use an ODBC interface, your system administrator must first install the appropriate ODBC driver on your computer.

Package - A method of encapsulating and storing related procedures, functions, variables, and other package constructs together as a unit in the database.

PL/SQL - Oracle’s procedural language extension to SQL.

Primary Key - The column that identifies or contributes to the identification of a unique instance of data in a module. The column or set of columns included in the definition of a table’s PRIMARY KEY constraint. A primary key’s values uniquely identify the rows in a table. Only one primary key may be defined per table.

Procedures - A set of SQL and PL/SQL statements that are grouped together as a unit to solve a specific problem or perform a set of related tasks. A procedure is created and stored in compiled form in the database and can be executed by a user or a database application. Procedures and functions are identical except that functions always return a single value to the caller, while procedures do not return values to the caller.

Referenced Key - The unique key or primary key of the same or different table that is referenced by a foreign key.

RunTime Processing - Database processes that occur while you are entering or saving the information. Effectively, these processes happen immediately, as opposed to 'batch processing' which is reserved for certain times when batches of information are processed at one time.

Schema - A collection of logical data storage structures (schema objects).

Segment - 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. For example, each table’s data is stored in its own data segment, while each index’s data is stored in its own index segment. If the table or index is partitioned, each partition is stored in its own segment.

Server Batch Queue - Batch process that prints designated reports at a specified date and time.

Slave Printer - A printer directly connected to the terminal / computer. Only used with character mode terminals.

Structured Query Language (SQL) - A simple yet powerful database access language that is the standard language for relational database management systems.

Table - A relational database structure which is analogous to the entity of a data model.

Tablespace - Where data is logically stored in a database.

Transaction - A logical unit of work that comprises one or more SQL statements executed by a single user.

Unique Index - An index where no two rows of a table have duplicate values in the columns that define the index.

Unique Key - A column set of columns included in the definition of a UNIQUE constraint.

User-Accessible View - The view in the data dictionary that summarizes and displays the information stored in the base tables. These views decode the base table data into useful information, such as user or table names, using joins and WHERE clauses to simplify the information. Most users are given access to the views rather than the base tables.

User Defined Fields (UDF) - Fields defined by an organization to customize modules and screens.