
ORACLE
ACADEMY

Database Programming with SQL

13-1
Creating Tables

ORACLE ACADEMY

Database Schema Objects

- The main database object types are:
  - Table
  - Index
  - Constraint
  - View
  - Sequence
  - Synonym
- Some of these object types can exist independently and others can not.

Table Creation

- All data in a relational database is stored in tables.
- When creating a new table, use the following rules for table names and column names:
  - Must begin with a letter
  - Must be 1 to 30 characters long
  - Must contain only A - Z, a - z, 0 - 9, _ (underscore), $, and #
  - Must not duplicate the name of another object owned by the same user
  - Must not be an Oracle Server reserved word

Naming Conventions

- It is best to use descriptive names for tables and other database objects.
- If a table will store information about students, name it STUDENTS, not PEOPLE or CHILDREN.
- Table names are not case sensitive.
- For example, STUDENTS is treated the same as STuDents or students.

Naming Conventions

- Table names should be plural, for example STUDENTS, not student.
- Creating tables is part of SQL's data definition language (DDL).
- Other DDL statements used to set up, change, and remove data structures from tables include ALTER, DROP, RENAME, and TRUNCATE.

CREATE TABLE

- To create a new table, you must have the CREATE TABLE privilege and a storage area for it.
- The database administrator uses data control language (DCL) statements to grant this privilege to users and assign a storage area.
- Tables belonging to other users are not in your schema.
- If you want to use a table that is not in your schema, use the table owner's name as a prefix to the table name:

```
SELECT *
FROM mary.students;
```

  - You must be granted access to a table to be able to select from it.

CREATE TABLE Syntax

- To create a new table, use the following syntax details:
  - table is the name of the table
  - column is the name of the column
  - Data type is the column's data type and length
  - DEFAULT expression specifies a default value if a value is omitted in the INSERT statement

```
CREATE TABLE table
(column data type [DEFAULT expression],
(column data type [DEFAULT expression],
(……[ ] );
```

CREATE TABLE Example

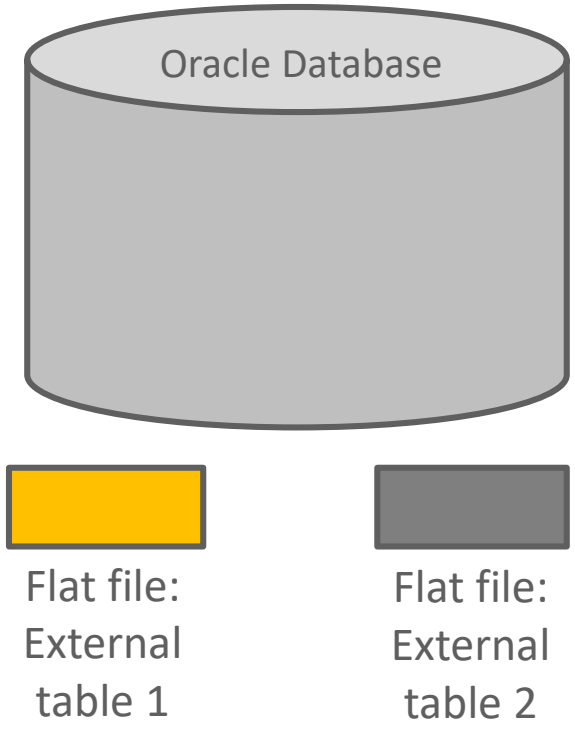
- The examples below show the CREATE TABLE statement:

```
CREATE TABLE my_cd_collection
 (cd_number NUMBER(3),
  title VARCHAR2(20),
  artist VARCHAR2(20),
  purchase_date DATE DEFAULT SYSDATE);
```

```
CREATE TABLE my_friends
 (first_name VARCHAR2(20),
  last_name VARCHAR2(30),
  email VARCHAR2(30),
  phone_num VARCHAR2(12),
  birth_date DATE);
```

External Tables

- Oracle also supports another table type: External table.
- In an external table, the data rows are not held inside the database files but are instead found in a flat file, stored externally to the database.
- Typically an external table is used to store data migrated from older versions of the databases used by a company.



External Tables

- The new syntax (shown in red) on the following slides, is not used in standard SQL statements for table creation.
- ORGANIZATION EXTERNAL -- tells Oracle to create an external table
- TYPE ORACLE_LOADER -- of type Oracle Loader (an Oracle Product)
- DEFAULT DIRECTORY def_dir1 -- the name of the directory for the file

External Tables

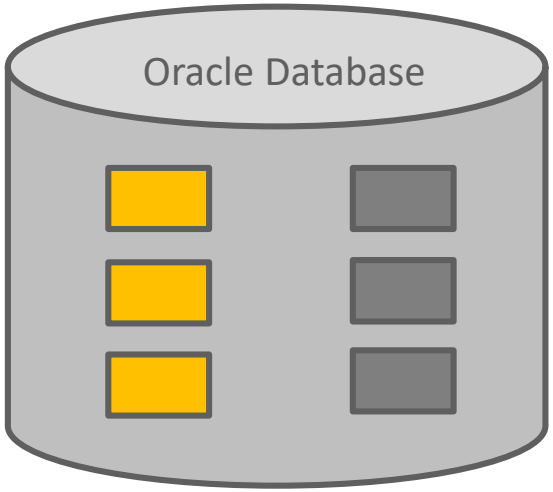
- ACCESS PARAMETERS -- how to read the file
- RECORDS DELIMITED BY NEWLINE -- how to identify the start of a new row
- FIELDS – the field name and data type specifications
- LOCATION – name of the actual file containing the data
- An example of the new syntax is found in red on the next slide.

External Tables Example

```
CREATE TABLE emp_load
  (employee_number CHAR(5),
   employee_dob CHAR(20),
   employee_last_name CHAR(20),
   employee_first_name CHAR(15),
   employee_middle_name CHAR(15),
   employee_hire_date DATE)
ORGANIZATION EXTERNAL
   (TYPE ORACLE_LOADER
    DEFAULT DIRECTORY def_dir1
    ACCESS PARAMETERS
   (RECORDS DELIMITED BY NEWLINE
    FIELDS (employee_number CHAR(2),
                    employee_dob CHAR(20),
                    employee_last_name CHAR(18),
                    employee_first_name CHAR(11),
                    employee_middle_name CHAR(11),
                    employee_hire_date CHAR(10) date_format DATE mask
"mm/dd/yyyy"))
   LOCATION ('info.dat'));
```

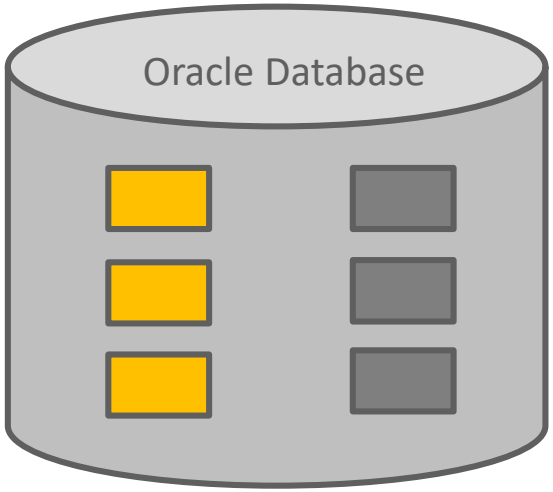
Data Dictionary

- Two kinds of tables exists in an Oracle Database: User tables and Data Dictionary tables.
- You can issue SQL statements to access both kinds of tables—you can select, insert, update, and delete data in the user tables, and you can select data in the Data Dictionary tables.



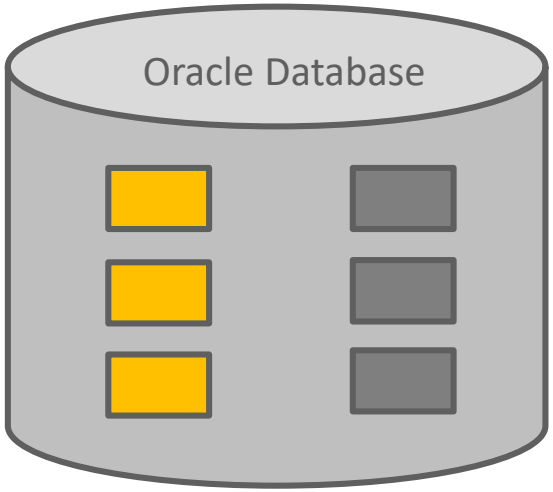
Data Dictionary

- User tables created by you containing your data:
  - employees, departments, jobs, etc.
- Data Dictionary tables:
  - DICTIONARY, USER_OBJECTS, USER_TABLES, USER_SEGMENTS, USER_INDEXES, etc.



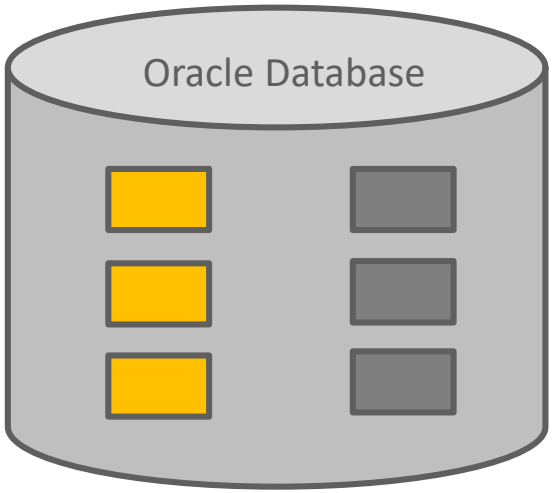
Data Dictionary

- The Data Dictionary tables are all owned by a special Oracle user called SYS and only SELECT statements should be used when working with any of these tables.
- To make these tables safe from accidental user access, they all have views created through which the Data Dictionary is accessed by database users.



Data Dictionary

- If any Oracle user attempts to do inserts, updates, or deletes against any of the Data Dictionary tables, the operation is disallowed as it might compromise the integrity of the entire database.



Data Dictionary

- When you are using the Data Dictionary views in the SQL Commands interface, you need to know the names of the Dictionary views you are working with.
- In Oracle, this is quite simple: prefix the object type you are looking for with a USER_xxx or an ALL_xxx, where xxx is the object type.

```
SELECT table_name, status
FROM USER_TABLES;
```

```
SELECT table_name, status
FROM ALL_TABLES;
```

Data Dictionary

- So if you want to investigate indexes, then simply select from USER_INDEXES; if you want information about sequences, then the table is USER_SEQUENCES and so on.

```
SELECT *
FROM user_indexes;
```

```
SELECT *
FROM user_objects
WHERE object_type = 'SEQUENCE';
```

Database Programming with SQL

13-2
Using Data Types

ORACLE ACADEMY

Common Data Types

- The most commonly used column data types for character and number values are below.
- For character values:
  - CHAR (fixed size, maximum 2000 characters)
  - VARCHAR2 (variable size, maximum 4000 characters)
  - CLOB (variable size, maximum 128 terabytes)
- For number values:
  - NUMBER (variable size, maximum precision 38 digits)

Common Data Types

- The most commonly used column data types for date, time, and binary values are below.
- For date and time values:
  - DATE
  - TIMESTAMP ….
  - INTERVAL
- For binary values (eg. multimedia: JPG, WAV, MP3, and so on):
  - RAW (variable size, maximum 2000 bytes)
  - BLOB (variable size, maximum 128 terabytes)

Common Data Types

- For character values, it is usually better to use VARCHAR2 or CLOB than CHAR, because it saves space.
- For example, an employee's last name is 'Chang'.
- In a VARCHAR2(30) column, only the 5 significant characters are stored: C h a n g.
- But in a CHAR(30) column, 25 trailing spaces would be stored as well, to make a fixed size of 30 characters.
- Number values can be negative as well as positive. For example, NUMBER(6,2) can store any value from +9999.99 down to –9999.99.

DATE-TIME Data Types

- The DATE data type stores a value of centuries down to whole seconds, but cannot store fractions of a second.
- '21-Aug-2003 17:25:30' is a valid value, but '21-Aug-2003 17:25:30.255' is not.
- The TIMESTAMP data type is an extension of the DATE data type which allows fractions of a second.
- For example, TIMESTAMP(3) allows 3 digits after the whole seconds, allowing values down to milliseconds to be stored.

DATE-TIME Data Types

- TIMESTAMP example:

```
CREATE TABLE time_ex1
(exact_time TIMESTAMP);
```

```
INSERT INTO time_ex1
VALUES ('10-Jun-2017 10:52:29.123456');
```

```
INSERT INTO time_ex1
VALUES (SYSDATE);
```

```
INSERT INTO time_ex1
VALUES (SYSTIMESTAMP);
```

```
SELECT *
FROM time_ex1;
```

EXACT_TIME
10-JUN-15 10.52.29.123456 AM
16-JUL-15 08.17.08.000000 AM
16-JUL-15 08.17.16.610293 AM

TIMESTAMP…With [LOCAL] Time Zone

- Think about the time value '17:30'. Of course it means "half past five in the afternoon".
- But in which time zone?
- Is it half past five New York City time or Beijing time or Istanbul time or …. ?
- In today's globalized organizations which operate in many different countries, it is important to know to which time zone a date-time value refers.

TIMESTAMP…With [LOCAL] Time Zone

- TIMESTAMP WITH TIME ZONE stores a time zone value as a displacement from Universal Coordinated Time or UCT (previously known as Greenwich Mean Time or GMT).
- A value of '21-Aug-2003 08:00:00 –5:00' means 8:00 am 5 hours behind UTC.
- This is US Eastern Standard Time (EST).

TIMESTAMP…With [LOCAL] Time Zone

- TIMESTAMP WITH TIME ZONE example:

```
CREATE TABLE time_ex2
(time_with_offset TIMESTAMP WITH TIME ZONE);
```

```
INSERT INTO time_ex2
VALUES (SYSTIMESTAMP);
```

```
INSERT INTO time_ex2
VALUES ('10-Jun-2017 10:52:29.123456 AM +2:00');
```

```
SELECT *
FROM time_ex2;
```

TIME_WITH_OFFSET
16-JUL-15 08.49.47.126056 AM -07:00
10-JUN-15 10.52.29.123456 AM +02:00

TIMESTAMP…With [LOCAL] Time Zone

- TIMESTAMP WITH LOCAL TIME ZONE is similar, but with one difference: when this column is selected in a SQL statement, the time is automatically converted to the selecting user's time zone.
- TIMESTAMP With…Time Zone Example:

```
CREATE TABLE time_ex3
( first_column TIMESTAMP WITH TIME ZONE,
  second_column TIMESTAMP WITH LOCAL TIME ZONE);
```

```
INSERT INTO time_ex3
  (first_column, second_column)
VALUES
  ('15-Jul-2017 08:00:00 AM -07:00', '15-Nov-2007 08:00:00');
```

TIMESTAMP…With Time Zone Example

- Both values are stored with a time displacement of –07:00 hours (PST).
- But now a user in Istanbul executes:

```
SELECT *
FROM time_ex3;
```

FIRST_COLUMN	SECOND_COLUMN
15-JUL-15 08.00.00.000000 AM -07:00	15-NOV-07 05.00.00.000000 PM

- Istanbul time is 9 hours ahead of PST; when it's 8am in Los Angeles, it's 5pm in Istanbul.

INTERVAL Data Types

- These store the elapsed time, or interval of time, between two date-time values.
- INTERVAL YEAR TO MONTH stores a period of time measured in years and months.
- INTERVAL DAY TO SECOND stores a period of time measured in days, hours, minutes, and seconds.

INTERVAL YEAR…TO MONTH

- Syntax:

```
INTERVAL YEAR [(year_precision)] TO MONTH
```

- year_precision is the maximum number of digits in the YEAR element.
- The default value of year_precision is 2.

INTERVAL YEAR…TO MONTH

- This example shows INTERVAL YEAR TO MONTH:

```
CREATE TABLE time_ex4
(loan_duration1 INTERVAL YEAR(3) TO MONTH,
 loan_duration2 INTERVAL YEAR(2) TO MONTH);
```

```
INSERT INTO time_ex4 (loan_duration1, loan_duration2)
VALUES (INTERVAL '120' MONTH(3),
         INTERVAL '3-6' YEAR TO MONTH);
```

Assume today's date is: 17-Jul-2017

```
SELECT SYSDATE + loan_duration1 AS "120 months from now",
       SYSDATE + loan_duration2 AS "3 years 6 months from now"
FROM time_ex4;
```

120 months from now	3 years 6 months from now
17-Jul-2027	17-Jan-2021

INTERVAL DAY…TO SECOND

- Use this when you need a more precise difference between two date-time values.
- Syntax:

```
INTERVAL DAY [day_precision)] TO SECOND [(fractional_seconds_precision)]
```

- day_precision is the maximum number of digits in the DAY element.
- The default value of day_precision is 2.
- fractional_seconds_precision is the number of digits in the fractional part of the SECOND date-time field.
- The default is 6.

INTERVAL DAY…TO SECOND

- This example shows interval DAY TO SECOND:

```
CREATE TABLE time_ex5
(day_duration1 INTERVAL DAY(3) TO SECOND,
 day_duration2 INTERVAL DAY(3) TO SECOND);
```

```
INSERT INTO time_ex5 (day_duration1, day_duration2)
VALUES (INTERVAL '25' DAY(2), INTERVAL '4 10:30:10' DAY TO SECOND);
```

```
SELECT SYSDATE + day_duration1 AS  "25 Days from now",
   TO_CHAR(SYSDATE + day_duration2, 'dd-Mon-yyyy hh:mi:ss')
   AS  "precise days and time from now"
FROM time_ex5;
```

25 Days from now	precise days and time from now
11-Aug-2017	21-Jul-2017 01:13:17

Database Programming with SQL

13-3
Modifying a Table

ORACLE ACADEMY

ALTER TABLE

- ALTER TABLE statements are used to:
  - Add a new column
  - Modify an existing column
  - Define a DEFAULT value for a column
  - Drop a column
- You can add or modify a column in a table, but you cannot specify where the column appears.

ALTER TABLE

- A newly added column always becomes the last column of the table.
- Also, if a table already has rows of data and you add a new column to the table, the new column is initially null for all of the pre-existing the rows.

ALTER TABLE: Adding a Column

- To add a new column, use the SQL syntax shown:

```
ALTER TABLE tablename
ADD (column name data type [DEFAULT expression],
column name data type [DEFAULT expression], ...
```

- For example:

```
ALTER TABLE my_cd_collection
ADD (release_date DATE DEFAULT SYSDATE);
```

```
ALTER TABLE my_friends
ADD (favorite_game VARCHAR2(30));
```

ALTER TABLE: Modifying a Column

- Modifying a column can include changes to a column's data type, size, and DEFAULT value.
- Rules and restrictions when modifying a column are:
  - You can increase the width or precision of a numeric column.
  - You can increase the width of a character column.
  - You can decrease the width of a NUMBER column if the column contains only null values, or if the table has no rows.
  - For VARCHAR types, you can decrease the width down to the largest value contained in the column.

ALTER TABLE: Modifying a Column

- You can change the data type only if the column contains null values.
- You can convert a CHAR column to VARCHAR2 or convert a VARCHAR2 COLUMN to CHAR only if the column contains null values, or if you do not change the size to something smaller than any value in the column.
- A change to the DEFAULT value of a column affects only later insertions to the table.

ALTER TABLE: Modifying a Column Example

- Example: a table has been created with two columns:

```
CREATE TABLE mod_emp
  (last_name VARCHAR2(20),
   salary NUMBER(8,2));
```

- Which of these modification would be allowed, and which would not? (Consider your answers both with and without rows of data in the table.)

```
ALTER TABLE mod_emp
     MODIFY (last_name VARCHAR2(30));
```

```
ALTER TABLE mod_emp
     MODIFY (last_name VARCHAR2(10));
```

```
ALTER TABLE mod_emp
     MODIFY (salary NUMBER(10,2));
```

```
ALTER TABLE mod_emp
     MODIFY (salary NUMBER(8,2) DEFAULT 50);
```

ALTER TABLE: Modifying a Column Example

- Would be permitted only if columns were empty, or the largest name was 10 or less characters

```
ALTER TABLE mod_emp
     MODIFY (last_name VARCHAR2(10));
```

- Would be permitted with or without data as column width increased.

```
ALTER TABLE mod_emp
     MODIFY (last_name VARCHAR2(30));
```

ALTER TABLE: Modifying a Column Example

```
ALTER TABLE mod_emp
     MODIFY (salary NUMBER(10,2));
```

- Would be permitted with or without data as column precision increased.

```
ALTER TABLE mod_emp
     MODIFY (salary NUMBER(8,2) DEFAULT 50);
```

- Would be permitted with or without data as only a DEFAULT value added.

ALTER TABLE: Dropping a Column

- When dropping a column the following rules apply:
  - A column containing data may be dropped.
  - Only one column can be dropped at a time.
  - You can't drop all of the columns in a table; at least one column must remain.
  - Once a column is dropped, the data values in it cannot be recovered.

ALTER TABLE: Dropping a Column

- SQL Syntax:

```
ALTER TABLE tablename
DROP COLUMN column name;
```

- For Example:

```
ALTER TABLE my_cd_collection
DROP COLUMN release_date;
```

```
ALTER TABLE my_friends
DROP COLUMN favorite_game;
```

SET UNUSED Columns

- Dropping a column from a large table can take a long time.
- A quicker alternative is to mark the column as unusable.
- The column values remain in the database but cannot be accessed in any way, so the effect is the same as dropping the column.
- In fact, you could add a new column to the database with the same name as the unused column.
- The unused columns are there, but invisible!
- Syntax:

```
ALTER TABLE tablename SET UNUSED (column name);
```

SET UNUSED Columns Example

- Example:

```
ALTER TABLE copy_employees
   SET UNUSED (email);
```

- DROP UNUSED COLUMNS removes all columns currently marked as unused.
- You use this statement when you want to reclaim the extra disk space from unused columns in a table.
- Example:

```
ALTER TABLE copy_employees
   DROP UNUSED COLUMNS;
```

ALTER TABLE Summarized

- This chart summarizes the uses of the ALTER TABLE command:

Syntax	Outcomes	Concerns
ALTER TABLE tablename ADD (column name data type [DEFAULT expression], column name data type [DEFAULT expression], …	Adds a new column to a table	You cannot specify where the column is to appear in the table. It becomes the last column.
ALTER TABLE tablename MODIFY (column name data type [DEFAULT expression], column name data type, …	Used to change a column's data type, size, and default value	A change to the default value of a column affects only subsequent insertions to the table.
ALTER TABLE tablename DROP COLUMN column name;	Used to drop a column from a table	The table must have at least one column remaining in it after it is altered. Once dropped, the column cannot be recovered.
ALTER TABLE tablename SET UNUSED (column name);	Used to mark one or more columns so they can be dropped later	Does not restore disk space. Columns are treated as if they were dropped.
ALTER TABLE tablename DROP UNUSED COLUMNS	Removes from the table all columns currently marked as unused	Once set unused, there is no access to the columns; no data displayed using DESCRIBE. Permanent removal; no rollback.

DROP TABLE

- The DROP TABLE statement removes the definition of an Oracle table.
- The database loses all the data in the table and all the indexes associated with it.
- When a DROP TABLE statement is issued:
  - All data is deleted from the table.
  - The table's description is removed from the Data Dictionary.
- The Oracle Server does not question your decision and it drops the table immediately.

DROP TABLE

- In the next slide, you will see that you may be able to restore a table after it is dropped, but it is not guaranteed.
- Only the creator of the table or a user with DROP ANY TABLE privilege (usually only the DBA) can remove a table.
- Syntax:

```
DROP TABLE tablename;
```

- Example:

```
DROP TABLE copy_employees;
```

FLASHBACK TABLE

- If you drop a table by mistake, you may be able to bring that table and its data back.
- Each database user has his own recycle bin into which dropped objects are moved, and they can be recovered from here with the FLASHBACK TABLE command.
- This command can be used to restore a table, a view, or an index that was dropped in error.
- The Syntax is:

```
FLASHBACK TABLE tablename TO BEFORE DROP;
```

FLASHBACK TABLE

- For example, if you drop the EMPLOYEES table in error, you can restore it by simply issuing the command:

```
FLASHBACK TABLE copy_employees TO BEFORE DROP;
```

- As the owner of a table, you can issue the flashback command, and if the table that you are restoring had any indexes, then these are also restored.
- It is possible to see which objects can be restored by querying the data dictionary view USER_RECYCLEBIN.

FLASHBACK TABLE

- The USER_RECYCLEBIN view can be queried like all other data dictionary views:

```
SELECT original_name, operation, droptime
FROM user_recyclebin
```

ORIGINAL_NAME	OPERATION	DROPTIME
EMPLOYEES	DROP	2007-12-05:12.34.24
EMP_PK	DROP	2007-12-05:12.34.24

FLASHBACK TABLE

- Once a table has been restored by the FLASHBACK TABLE command, it is no longer visible in the USER_RECYCLEBIN view.
- Any indexes that were dropped with the original table will also be restored.
- It may be necessary (for security reasons) to completely drop a table, bypassing the recycle bin.
- This can be done by adding the keyword PURGE.

```
DROP TABLE copy_employees PURGE;
```

RENAME

- To change the name of a table, use the RENAME statement.
- This can be done only by the owner of the object or by the DBA.
- Syntax:

```
RENAME old_name to new_name;
```

- Example:

```
RENAME my_cd_collection TO my_music;
```

- We will see later that we can rename other types of objects such as views, sequences, and synonyms.

TRUNCATE

- Truncating a table removes all rows from a table and releases the storage space used by that table.
- When using the TRUNCATE TABLE statement:
  - You cannot roll back row removal.
  - You must be the owner of the table or have been given DROP ANY TABLE system privileges.

TRUNCATE

- Syntax:

```
TRUNCATE TABLE tablename;
```

- The DELETE statement also removes rows from a table, but it does not release storage space.
- TRUNCATE is faster than DELETE because it does not generate rollback information.

COMMENT ON TABLE

- You can add a comment of up to 2,000 characters about a column, table, or view by using the COMMENT statement.
- Syntax:

```
COMMENT ON TABLE tablename | COLUMN table.column
     IS 'place your comment here';
```

- Example:

```
COMMENT ON TABLE employees
     IS 'Western Region only';
```

COMMENT ON TABLE

- To view the table comments in the data dictionary:

```
SELECT table_name, comments
FROM user_tab_comments;
```

TABLE_NAME	COMMENTS
EMPLOYEES	Western Region Only

- If you want to drop a comment previously made on a table or column, use the empty string(''):

```
COMMENT ON TABLE employees IS ' ' ;
```

FLASHBACK QUERY

- You may discover that data in a table has somehow been inappropriately changed.
- Luckily, Oracle has a facility that allows you to view row data at specific points in time, so you can compare different versions of a row over time.
- This facility is very useful.
- Imagine, for instance, that someone accidently performs some DML on a table, and then executes a COMMIT on those changes.
- Oracle Application Express commits automatically, so mistakes are easily made.

FLASHBACK QUERY

- You can use the FLASHBACK QUERY facility to examine what the rows looked like BEFORE those changes were applied.
- When Oracle changes data, it always keeps a copy of what the amended data looked like before any changes were made.
- So it keeps a copy of the old column value for a column update, it keeps the entire row for a delete, and it keeps nothing for an insert statement.

FLASHBACK QUERY

- These old copies are held in a special place called the UNDO tablespace.
- Users can access this special area of the Database using a flashback query.
- You can look at older versions of data by using the VERSIONS clause in a SELECT statement.

FLASHBACK QUERY

- For example:

```
SELECT employee_id,first_name ||' '|| last_name AS "NAME",
    versions_operation AS "OPERATION",
    versions_starttime AS "START_DATE",
    versions_endtime  AS "END_DATE", salary
FROM employees
  VERSIONS BETWEEN SCN MINVALUE AND MAXVALUE
WHERE  employee_id = 1;
```

- The SCN number referred to in the above query means the System Change Number and is a precise identification of time in the database.
- It is a sequential number incremented and maintained by the database itself.

FLASHBACK QUERY

- The best way to demonstrate FLASHBACK QUERY is with an example.
- The contents are as follows for employee_id 1 in the employees table.

```
SELECT employee_id,first_name ||' '|| last_name AS "NAME",
               versions_operation AS "OPERATION",
               versions_starttime AS "START_DATE",
               versions_endtime  AS "END_DATE", salary
FROM copy_employees
  VERSIONS BETWEEN SCN MINVALUE AND MAXVALUE
WHERE  employee_id = 1;
```

no data found

FLASHBACK QUERY

- Then we create the employee:

```
INSERT INTO copy_employees
VALUES (1, 'Natacha', 'Hansen', 'NHANSEN', '4412312341234',
        '07-SEP-1998', 'AD_VP', 12000, null, 100, 90, NULL);
```

```
SELECT employee_id,first_name ||' '|| last_name AS "NAME",
               versions_operation AS "OPERATION",
               versions_starttime AS "START_DATE",
               versions_endtime  AS "END_DATE", salary
FROM copy_employees
  VERSIONS BETWEEN SCN MINVALUE AND MAXVALUE
WHERE  employee_id = 1;
```

EMPLOYEE_ID	NAME	OPERATION	START_DATE	END_DATE	SALARY
1	Natacha Hansen	I	07-SEP-1998 06.51.58 AM	-	12000

FLASHBACK QUERY

- Then you can update the row:

```
UPDATE copy_employees
SET salary = 1
WHERE employee_id = 1;
```

```
SELECT employee_id,first_name ||' '|| last_name AS "NAME",
               versions_operation AS "OPERATION",
               versions_starttime AS "START_DATE",
               versions_endtime  AS "END_DATE", salary
FROM copy_employees
  VERSIONS BETWEEN SCN MINVALUE AND MAXVALUE
WHERE  employee_id = 1;
```

EMPLOYEE_ID	NAME	OPERATION	START_DATE	END_DATE	SALARY
1	Natacha Hansen	U	07-SEP-1998 06.57.01 AM	-	1
1	Natacha Hansen	I	07-SEP-1998 06.51.58 AM	07-SEP-1998 06.57.01 AM	12000

FLASHBACK QUERY

- Then you can delete the row:

```
DELETE from copy_employees
WHERE employee_id = 1;
```

```
SELECT employee_id,first_name ||' '|| last_name AS "NAME",
                versions_operation AS "OPERATION",
                versions_starttime AS "START_DATE",
                versions_endtime  AS "END_DATE", salary
FROM copy_employees
  VERSIONS BETWEEN SCN MINVALUE AND MAXVALUE
WHERE  employee_id = 1;
```

EMPLOYEE_ID	NAME	OPERATION	START_DATE	END_DATE	SALARY
1	Natacha Hansen	D	07-SEP-1998 07.00.10 AM	-	1
1	Natacha Hansen	U	07-SEP-1998 06.57.01 AM	07-SEP-1998 07.00.10 AM	1
1	Natacha Hansen	I	07-SEP-1998 06.51.58 AM	07-SEP-1998 06.57.01 AM	12000

FLASHBACK QUERY

- The result from the last query on the previous slide is only available when using Flashback query, i.e. the VERSIONS clause.
- If you attempt a normal search from employee_id = 1 following the delete statement, you would have received the normal error, No Data Found.

```
SELECT employee_id, salary
FROM copy_employees
WHERE employee_id = 1;
```

Database Programming with SQL

14-1
Intro to Constraints; NOT NULL and UNIQUE Constraints

ORACLE ACADEMY

Constraints at the Column Level

- A column-level constraint references a single column.
- To establish a column-level constraint, the constraint must be defined in the CREATE TABLE statement as part of the column definition.
- Examine the following SQL statement that establishes a column-level constraint.

```
CREATE TABLE clients
(client_number NUMBER(4) CONSTRAINT clients_client_num_pk PRIMARY KEY,
 first_name        VARCHAR2(14),
 last_name        VARCHAR2(13));
```

Constraints at the Column Level

- CREATE TABLE clients

```
CREATE TABLE clients
(client_number NUMBER(4) CONSTRAINT clients_client_num_pk PRIMARY KEY,
 first_name        VARCHAR2(14),
 last_name        VARCHAR2(13));
```

- The name of the constraint is clients_client_num_pk.
- It enforces the business rule that the client_number is the primary key of the clients table.

Constraints at the Table Level

- Table-level constraints are listed separately from the column definitions in the CREATE TABLE statement.
- Table-level constraint definitions are listed after all the table columns have been defined.
- In the example shown, the unique constraint is listed last in the CREATE TABLE statement.

```
CREATE TABLE clients (
 client_number NUMBER(6) NOT NULL,
 first_name    VARCHAR2(20),
 last_name     VARCHAR2(20),
 phone         VARCHAR2(20),
 email         VARCHAR2(10) NOT NULL,
CONSTRAINT clients_phone_email_uk UNIQUE (email,phone));
```

Basic Rules for Constraints

- Constraints that refer to more than one column (a composite key) must be defined at the table level
- The NOT NULL constraint can be specified only at the column level, not the table level
- UNIQUE, PRIMARY KEY, FOREIGN KEY, and CHECK constraints can be defined at either the column or table level
- If the word CONSTRAINT is used in a CREATE TABLE statement, you must give the constraint a name

Examine the Violations

```
CREATE TABLE clients(
        client_number  NUMBER(6),
        first_name     VARCHAR2(20),
        last_name      VARCHAR2(20),
        phone          VARCHAR2(20) CONSTRAINT phone_email_uk
                              UNIQUE(email,phone),
         email         VARCHAR2(10) CONSTRAINT NOT NULL,
     CONSTRAINT emailclients_email NOT NULL,
     CONSTRAINT clients_client_num_pk PRIMARY KEY (client_number));
```

Examine the Violations

COMPOSITE UNIQUE KEY VIOLATION
Composite keys must be defined at the table level.

```
CREATE TABLE clients(
        client_number   NUMBER(6),
        first_name      VARCHAR2(20),
        last_name       VARCHAR2(20),
        phone           VARCHAR2(20) CONSTRAINT phone_email_uk
                                UNIQUE(email,phone),
         email          VARCHAR2(10) CONSTRAINT NOT NULL,
     CONSTRAINT emailclients_email NOT NULL,
     CONSTRAINT clients_client_num_pk PRIMARY KEY (client_number));
```

NOT NULL VIOLATION
NOT NULL constraints can only be defined at the column level.

NAME VIOLATION
When using the term CONSTRAINT, it must be followed by a constraint name.

Five Types of Constraints

- Five types of constraints exist within an Oracle database.
- Each type enforces a different rule.
- The types are:
  - NOT NULL constraints
  - UNIQUE constraints
  - PRIMARY KEY constraints
  - FOREIGN KEY constraints
  - CHECK constraints

Database Programming with SQL

14-2
PRIMARY KEY, FOREIGN KEY, and CHECK Constraints

ORACLE ACADEMY

PRIMARY KEY Constraints

- In a CREATE TABLE statement, the column-level PRIMARY KEY constraint syntax is stated:

```
CREATE TABLE clients
(client_number NUMBER(4) CONSTRAINT clients_client_num_pk PRIMARY KEY,
first_name VARCHAR2(14),
last_name VARCHAR2(13));
```

- Note that the column-level simply refers to the area in the CREATE TABLE statement where the columns are defined.
- The table level refers to the last line in the statement below the list of individual column names.

PRIMARY KEY Constraints

- To create the PRIMARY KEY constraint at table-level the syntax is:

```
CREATE TABLE clients
(client_number NUMBER(4),
 first_name VARCHAR2(14),
 last_name VARCHAR2(13),
CONSTRAINT clients_client_num_pk PRIMARY KEY (client_number));
```

- Note that the PRIMARY KEY column name follows the constraint type, and is enclosed in parenthesis.

PRIMARY KEY Constraints

- To define a composite PRIMARY KEY, you must define the constraint at the table level rather than the column level.
- An example of a composite primary key constraint is shown below:

```
CREATE TABLE copy_job_history
(employee_id NUMBER(6,0),
 start_date DATE,
 job_id VARCHAR2(10),
 department_id NUMBER(4,0),
CONSTRAINT copy_jhist_id_st_date_pk PRIMARY KEY(employee_id, start_date));
```

FOREIGN KEY (REFERENTIAL INTEGRITY) Constraints

- FOREIGN KEY constraints are also called "referential integrity" constraints.
- Foreign Key constraints designate a column or combination of columns as a foreign key.
- A foreign keys links back to the primary key (or a unique key) in another table, and this link is the basis of the relationship between tables.

FOREIGN KEY Constraint Syntax

- The syntax for defining a FOREIGN KEY constraint requires a reference to the table and column in the parent table.
- A FOREIGN KEY constraint in a CREATE TABLE statement can be defined as follows.
- Column-level syntax example:

```
CREATE TABLE copy_employees
(employee_id NUMBER(6,0) CONSTRAINT copy_emp_pk PRIMARY KEY,
 first_name VARCHAR2(20),
 last_name VARCHAR2(25),
 department_id NUMBER(4,0) CONSTRAINT c_emps_dept_id_fk
                           REFERENCES departments(department_id),
 email VARCHAR2(25));
```

FOREIGN KEY Constraint Syntax

- The syntax for defining a FOREIGN KEY constraint requires a reference to the table and column in the parent table.
- A FOREIGN KEY constraint in a CREATE TABLE statement can be defined as follows.
- Table-level syntax example:

```
CREATE TABLE copy_employees
(employee_id NUMBER(6,0) CONSTRAINT copy_emp_pk PRIMARY KEY,
 first_name VARCHAR2(20),
 last_name VARCHAR2(25),
 department_id NUMBER(4,0),
 email VARCHAR2(25),
 CONSTRAINT c_emps_dept_id_fk FOREIGN KEY (department_id)
                              REFERENCES departments(department_id));
```

ON DELETE CASCADE - Maintaining Referential Integrity

- Using the ON DELETE CASCADE option when defining a foreign key enables the dependent rows in the child table to be deleted when a row in the parent table is deleted.
- If the foreign key does not have an ON DELETE CASCADE option, referenced rows in the parent table cannot be deleted.
- In other words, the child table FOREIGN KEY constraint includes the ON DELETE CASCADE permission allowing its parent to delete the rows that it refers to.

ON DELETE CASCADE Syntax

- Table created without ON DELETE CASCADE:

```
CREATE TABLE copy_employees
(employee_id NUMBER(6,0) CONSTRAINT copy_emp_pk PRIMARY KEY,
 first_name VARCHAR2(20),
 last_name VARCHAR2(25),
 department_id NUMBER(4,0),
 email VARCHAR2(25),
 CONSTRAINT cdept_dept_id_fk FOREIGN KEY (department_id)
        REFERENCES copy_departments(department_id));
```

- An attempt to delete department_id 110 from the departments table fails as there are dependent rows in the employee table.

```
ORA-02292: integrity constraint (US_A009EMEA815_PLSQL_T01.CDEPT_DEPT_ID_FK)
violated - child record found
```

ON DELETE CASCADE Syntax

- Table created with ON DELETE CASCADE:

```
CREATE TABLE copy_employees
(employee_id NUMBER(6,0) CONSTRAINT copy_emp_pk PRIMARY KEY,
 first_name VARCHAR2(20),
 last_name VARCHAR2(25),
 department_id NUMBER(4,0),
 email VARCHAR2(25),
 CONSTRAINT cdept_dept_id_fk FOREIGN KEY (department_id)
    REFERENCES copy_departments(department_id) ON DELETE CASCADE);
```

- An attempt to delete department_id 110 from the departments table succeeds, and the dependent rows in the employee table are also deleted.
- 1 row(s) deleted.

ON DELETE SET NULL

- Rather than having the rows in the child table deleted when using an ON DELETE CASCADE option, the child rows can be filled with null values using the ON DELETE SET NULL option.

```
CREATE TABLE copy_employees
(employee_id NUMBER(6,0) CONSTRAINT copy_emp_pk PRIMARY KEY,
 first_name VARCHAR2(20),
 last_name VARCHAR2(25),
 department_id NUMBER(4,0),
 email VARCHAR2(25),
 CONSTRAINT cdept_dept_id_fk FOREIGN KEY (department_id)
    REFERENCES copy_departments(department_id) ON DELETE SET NULL);
```

CHECK Constraints

- The CHECK constraint explicitly defines a condition that must be met.
- To satisfy the constraint, each row in the table must make the condition either True or unknown (due to a null).
- The condition of a CHECK constraint can refer to any column in the specified table, but not to columns of other tables.

CHECK Constraint Example

- This CHECK constraint ensures that a value entered for end_date is later than start_date.

```
CREATE TABLE copy_job_history
(employee_id NUMBER(6,0),
 start_date DATE,
 end_date DATE,
 job_id VARCHAR2(10),
 department_id NUMBER(4,0),
 CONSTRAINT cjhist_emp_id_st_date_pk
           PRIMARY KEY(employee_id, start_date),
 CONSTRAINT cjhist_end_ck CHECK (end_date > start_date));
```

- As this CHECK CONSTRAINT is referencing two columns in the table, it MUST be defined at table level.

CHECK Constraint Conditions

- A CHECK constraint must only be on the row where the constraint is defined.
- A CHECK constraint cannot be used in queries that refer to values in other rows.
- The CHECK constraint cannot contain calls to the functions SYSDATE, UID, USER, or USERENV.
- The statement CHECK(SYSDATE >'05-May-1999') is not allowed.

CHECK Constraint Conditions

- The CHECK constraint cannot use the pseudocolumns CURRVAL, NEXTVAL, LEVEL, or ROWNUM.
- The statement CHECK(NEXTVAL > 0) is not allowed.
- A single column can have multiple CHECK constraints that reference the column in its definition.
- There is no limit to the number of CHECK constraints that you can define on a column.

CHECK Constraint Syntax

- CHECK constraints can be defined at the column level or the table level.
- The syntax to define a CHECK constraint is:
  - Column-level syntax:

```
salary NUMBER(8,2) CONSTRAINT employees_min_sal_ck CHECK (salary > 0)
```

  - Table-level syntax:

```
CONSTRAINT employees_min_sal_ck CHECK (salary > 0)
```

Database Programming with SQL

14-3
Managing Constraints

ORACLE ACADEMY

The ALTER Statement

- The ALTER statement requires:
  - name of the table
  - name of the constraint
  - type of constraint
  - name of the column affected by the constraint
- In the code example shown below, using the employees table, the primary-key constraint could have been added after the table was originally created.

```
ALTER TABLE employees
ADD CONSTRAINT  emp_id_pk PRIMARY KEY (employee_id);
```

Adding Constraints

- To add a constraint to an existing table, use the following SQL syntax:

```
ALTER TABLE table_name
ADD [CONSTRAINT constraint_name] type of constraint (column_name);
```

- If the constraint is a FOREIGN KEY constraint, the REFERENCES keyword must be included in the statement.
- Syntax:

```
ALTER TABLE tablename
ADD CONSTRAINT constraint_name FOREIGN KEY(column_name) REFERENCES
tablename(column_name);
```

Adding Constraints Conditions

- If the constraint is a NOT NULL constraint, the ALTER TABLE statement uses MODIFY in place of ADD.
- NOT NULL constraints can be added only if the table is empty or if the column contains a value for every row:

```
ALTER TABLE table_name
MODIFY (column_name CONSTRAINT constraint_name NOT NULL);
```

```
ALTER TABLE employees
MODIFY (email CONSTRAINT emp_email_nn NOT NULL);
```

Why Enable and Disable Constraints?

- To enforce the rules defined by integrity constraints, the constraints should always be enabled.
- In certain situations, however, it is desirable to temporarily disable the integrity constraints of a table for performance reasons, such as:
  - When loading large amounts of data into a table
  - When performing batch operations that make massive changes to a table (such as changing everyone's employee number by adding 1,000 to the existing number)

Dropping Constraints

- To drop a constraint, you need to know the name of the constraint.
- If you do not know it, you can find the constraint name from the USER_CONSTRAINTS and USER_CONS_COLUMNS in the data dictionary.
- The CASCADE option of the DROP clause causes any dependent constraints also to be dropped.
- Note that when you drop an integrity constraint, that constraint is no longer enforced by the Oracle Server and is no longer available in the data dictionary.

Dropping Constraints

- No rows or any data in any of the affected tables are deleted when you drop a constraint.

```
ALTER TABLE table_name
DROP CONSTRAINT  name [CASCADE}
```

```
ALTER TABLE copy_departments
DROP CONSTRAINT  c_dept_dept_id_pk CASCADE;
```

Using the DISABLE Clause

- You can use the DISABLE clause in both the ALTER TABLE statement and the CREATE TABLE statement.

```
CREATE TABLE copy_employees
( employee_id NUMBER(6,0) PRIMARY KEY DISABLE,
   ...
   ...);

ALTER TABLE copy_employees
DISABLE CONSTRAINT c_emp_dept_id_fk;
```

- Disabling a unique or primary-key constraint removes the unique index.

Using the CASCADE Clause

- The CASCADE clause disables dependent integrity constraints. If the constraint is later enabled, the dependent constraints are not automatically enabled.
- Syntax and example:

```
ALTER TABLE table_name
DISABLE CONSTRAINT constraint_name [CASCADE];
```

```
ALTER TABLE copy_departments
DISABLE CONSTRAINT c_dept_dept_id_pk CASCADE;;
```

Enabling Constraints

- To activate an integrity constraint currently disabled, use the ENABLE clause in the ALTER TABLE statement.
- ENABLE ensures that all incoming data conforms to the constraint.
- Syntax and example:

```
ALTER TABLE table_name
ENABLE CONSTRAINT constraint_name;
```

```
ALTER TABLE copy_departments
ENABLE CONSTRAINT c_dept_dept_id_pk;
```

- You can use the ENABLE clause in both the CREATE TABLE statement and the ALTER TABLE statement.

Enabling Constraint Considerations

- If you enable a constraint, that constraint applies to all the data in the table.
- All the data in the table must fit the constraint.
- If you enable a UNIQUE KEY or PRIMARY KEY constraint, a UNIQUE or PRIMARY KEY index is created automatically.
- Enabling a PRIMARY KEY constraint that was disabled with the CASCADE option does not enable any foreign keys that are dependent on the primary key.
- ENABLE switches the constraint back on after you switched it off.

Cascading Constraints

- If an ALTER TABLE statement does not include the CASCADE CONSTRAINTS option, any attempt to drop a primary key or multicolumn constraint will fail.
- Remember, you can't delete a parent value if child values exist in other tables.

```
ALTER TABLE table_name
DROP(column name(s)) CASCADE CONSTRAINTS;
```

When CASCADE is Not Required

- If all columns referenced by the constraints defined on the dropped columns are also dropped, then CASCADE CONSTRAINTS is not required.
- For example, assuming that no other referential constraints from other tables refer to column PK, it is valid to submit the following statement without the CASCADE CONSTRAINTS clause:

```
ALTER TABLE tablename DROP
(pk_column_name(s));
```

- However, if any constraint is referenced by columns from other tables or remaining columns in the target table, you must specify CASCADE CONSTRAINTS to avoid an error.

Viewing Constraints

- To view all constraints on your table, query the USER_CONSTRAINTS table.

```
SELECT constraint_name, table_name, constraint_type, status
FROM USER_CONSTRAINTS
WHERE table_name ='COPY_EMPLOYEES';
```

CONSTRAINT_NAME	TABLE_NAME	CONSTRAINT_TYPE	STATUS
COPY_EMP_PK	COPY_EMPLOYEES	P	ENABLED
CDEPT_DEPT_ID_FK	COPY_EMPLOYEES	R	ENABLED

Query USER_CONSTRAINTS

- The constraint types listed in the Data Dictionary are:
  - P – PRIMARY KEY; R – REFERENCES (foreign key);
  - C – CHECK constraint (including NOT NULL);
  - U – UNIQUE.

CONSTRAINT_NAME	TABLE_NAME	CONSTRAINT_TYPE	STATUS
COPY_EMP_PK	COPY_EMPLOYEES	P	ENABLED
CDEPT_DEPT_ID_FK	COPY_EMPLOYEES	R	ENABLED