



Applied Database Systems

Section 2

Database Basics – Part 2

Tables, Keys, SQL Basics

ORACLE
Academy



Copyright © 2022, Oracle and/or its affiliates. Oracle, Java, and MySQL are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Objectives

- This lesson covers the following objectives:
 - Database Refresh/SQL
 - Creating Databases, Inserting, and Updating Data
 - Retrieving/Conditionally Retrieving Data
 - Ordering Data, Using Indexes
 - Joining Tables
 - Lab 2



An abstract landscape illustration. The top half has a light green background with a sun in the top right corner, rendered as a yellow circle with radiating lines. To the left of the sun is a colorful, abstract mountain shape composed of various colored segments (purple, red, blue, black, orange). Further left is a white, stylized mountain range. The bottom half of the image features a green foreground with a complex, interlocking geometric pattern of lines. The title 'Database Background Refresh' is centered in the upper half of the image.

Database Background Refresh

ORACLE
Academy

So, What is a Database?



A **database** is a collection of data treated as a unit



A Database Management System (DBMS) is software that organizes data in a **database**



The purpose of a database is to store, manage and provide access to data



Data, DBMS and associated application are referred to as a database system

ORACLE
Academy

ADS – Section 2
Database Basics – Part 2

Copyright © 2022, Oracle and/or its affiliates. Oracle, Java, and MySQL are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

5

There is no value in data until you actually 'do' something with it. Databases make this data usable by providing access to it and giving you a place where that data can be stored.

Relational Databases - How Do Relational Databases Store Data?

- Data is stored in table format with rows and columns

| EMPNO | ENAME | DEPTNO | JOB | HIREDATE |
|-------|-------|--------|----------|-----------|
| 7876 | ADAMS | 20 | CLERK | 23-MAY-87 |
| 7499 | ALLEN | 30 | SALESREP | 20-FEB-81 |
| 7698 | BLAKE | 30 | MANAGER | 01-MAY-81 |
| 7782 | CLARK | 10 | MANAGER | 09-JUN-81 |
| 7902 | FORD | 20 | ANALYST | 03-DEC-81 |

↑
A **column** in a table
contains an attribute

← A **row** in a
table contains
transaction
information

How do relational databases store data?

- Data is stored in a table format with rows and columns
- A column in a table contains an attribute
- A row in a table contains a transaction or record; records consist of many attributes
- When you add data into a database, you typically add a row, in other words a new record



Let's Start With SQL

ORACLE
Academy

What Does SQL Stand For?

- SQL stands for Structured Query Language
- SQL is a well-defined standard language use for interacting with relation database engines



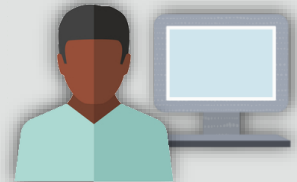
How To Use a Relational Database?

- In a relational database, you do not specify the access route to the tables, and you do not need to know how the data is arranged physically
- To access the database, you execute a SQL statement, which is the American National Standards Institute (ANSI) standard language for operating relational databases
- SQL is also compliant with ISO Standard (SQL 1999)



Execute SQL Statements to Work with a Database

- Used to access and describe the data stored in the database
- Define data stored in the database and manipulate that data
 - Creating, replacing, altering, and dropping database objects
 - Inserting, updating, and deleting rows in a table
 - Querying data stored in the database
 - Controlling access to the database and database objects
 - Guaranteeing database consistency and integrity



ORACLE
Academy

ADS – Section 2
Database Basics – Part 2

Copyright © 2022, Oracle and/or its affiliates. Oracle, Java, and MySQL are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

10

In a relational database, you specify the command in a logical way rather than in a direct storage access way. You execute SQL statements to work with a database.

A database needs to make sure that when you're running different operations and transactions, that the transaction has what's considered integrity, from beginning to end.

Types of SQL Statements

- **DDL** (Data Definition Language) – defines database structures
- **DML** (Data Manipulation Language) – manipulates data (INSERT, UPDATE, DELETE)
- Usually, subsets of
 - **DQL** (Data Query Language) – SELECTs data
 - **DCL** (Data Control Language) – controls user access
 - **TCL** (Transactional Control Language) – manages database transactions



Guidelines for Constructing Valid SQL Statements

- SQL statements are not case-sensitive (some databases implement case sensitivity)
- SQL statements can be entered on one or more lines
- SQL statements end with a semi-colon ;
- Keywords cannot be abbreviated or split across lines, typically spelled with uppercase letters
- Clauses are usually placed on separate lines
- Indents are used to enhance readability

```
CREATE TABLE dept (  
    deptno      NUMBER (2) ,  
    dname       VARCHAR2 (14) ,  
    loc         VARCHAR2 (13  
);
```

If you're in an operating system that is case sensitive, for example, Unix, there may be situations where the SQL statements and the object names are case sensitive. But that is very specific to implementations on specific operating systems, and particular databases.

An abstract landscape illustration. The top half has a light green background with a sun in the top right corner, rendered as a yellow circle with radiating lines. To the left of the sun is a mountain range with a peak covered in a colorful, patchwork pattern of purple, red, blue, and black. Below the sun and mountain are some white, wispy cloud-like shapes. The bottom half of the image features a green hill on the left and a larger hill on the right covered in a dense, repeating geometric pattern of small, interlocking shapes. The title "Creating a Database" is centered in the upper half of the image.

Creating a Database

ORACLE
Academy

Data Definition Language (DDL)

- DDL is used to create database objects

| Object | Description |
|-----------------|--|
| Table | Is the basic unit of storage; consists of rows |
| View | Logically represents subsets of data from one or more tables |
| Sequence | Generates numeric values |
| Index | Improves the performance of some queries |
| Synonym | Gives an alternative name to an object |

Tables and Columns

- Tables are the basic unit of data storage
- Data is stored in rows and columns
- Define a table with a table name, such as employees, and a set of columns
- Each column has a:
 - Column name, such as employee_id, last_name, and job_id
 - Datatype, such as VARCHAR2, DATE, or NUMBER
 - Width, if columns are of the NUMBER datatype, define precision and scale instead of width
- A row is a collection of column information corresponding to a single record

Relational Database Terminology

| 2 EMPLOYEE _ID | FIRST _NAME | LAST _NAME | 3 SALARY | COMMISSION _PCT | 4 DEPARTMENT _ID |
|----------------------|----------------|---------------|-------------|--------------------|------------------------|
| 100 | Steven | King | 24000 | - | 90 |
| 101 | Neena | Kochhar | 17000 | - | 90 |
| 102 | Lex | De Haan | 17000 | - | 90 |
| 200 | Jennifer | Whalen | 4400 | - | 10 |
| 205 | Shelley | Higgins | 12000 | - | 110 |
| 206 | William | Gietz | 8300 | - | 110 |
| 149 1 | Eleni | Zlotkey | 10500 | .2 | 80 |
| 174 | Ellen | Abel | 11000 | .3 | 80 |
| 201 | Michael | Hartstein | 13000 | - | 20 |

1. **Row:** representing all the data required for a particular employee
2. **Column:** represents one kind of data in a table, containing the primary key value
3. **Column:** represents one kind of data in a table, not a key value
4. **Column:** represents one kind of data in a table, containing the foreign key value
5. **Field:** found at the intersection of a row and a column; must be single-valued
6. **Empty field:** no value in it; referred to as a null value

ORACLE
Academy

ADS – Section 2
Database Basics – Part 2

Copyright © 2022, Oracle and/or its affiliates. Oracle, Java, and MySQL are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

16

Some Things to Know About Relational Models

- The order of the rows and columns is not important
- Every row is unique (unless duplicates specifically allowed)
- Each field can contain only one value
- Values within a column or field are from the same domain (datatype)
- Table names must be unique
- Column names within each table must be unique



Table Relationships

- Many tables in a database are related to other tables in a database
 - Each row of data in a table can be uniquely identified by a primary key
 - You can logically relate data from multiple tables using foreign keys

Table name: **EMPLOYEES**

| EMPLOYEE_ID | FIRST_NAME | LAST_NAME | DEPARTMENT_ID |
|-------------|------------|-----------|---------------|
| 100 | Steven | King | 90 |
| 101 | Neena | Kochhar | 90 |
| 103 | David | Austin | 60 |

↑ Primary key

↑ Foreign key

Table name: **DEPARTMENTS**

| DEPARTMENT_ID | DEPARTMENT_NAME | MANAGER_ID |
|---------------|-----------------|------------|
| 60 | IT | 103 |
| 90 | Executive | 100 |

↑ Primary key

ORACLE
Academy

ADS – Section 2
Database Basics – Part 2

Copyright © 2022, Oracle and/or its affiliates. Oracle, Java, and MySQL are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

18

Tables in a database are related by this combination here called primary key and foreign key. A table will have a primary key that will ensure that it is a unique column in that table, and it will advertise itself as a primary key. Any other table, in this case employees, that wants to find out what department IDs exist in the department table can say, I want to link my department ID column to the department ID column in departments. This occurs by calling my department ID column a foreign key that points to the department's table primary key.

Table Management

Data Definition Language (DDL)

- The most common table commands of DDL:
 - CREATE: to create a table
 - ALTER: to modify an object's structure
 - DROP: to remove an object from the database
 - RENAME: to rename a database object



Creating Tables: Naming Rules

- Table names and column names must:
 - Begin with a letter
 - Be 1–30 characters long
 - Contain only A–Z, a–z, 0–9, _, \$, and #
 - Not duplicate the name of another object owned by the same user
- Table names are not case-sensitive
 - For example, EMPLOYEES is treated the same as eMPloyees or eMpLOYEES



CREATE TABLE Statement

- Specify in the statement:
 - Table name
 - Column name, column data type, column size
 - Integrity constraints (optional)
 - Default values (optional)



```
CREATE TABLE [schema.]table  
    (column datatype [DEFAULT expr][, ...]);
```

The CREATE TABLE statement here creates a table and at a minimum you need to specify the table name and one column, including the type of column it is. There are other optional things you can specify about a table like constraints and default values.

Creating Tables

- Create the table:

```
CREATE TABLE dept(  
  deptno      NUMBER(2) ,  
  dname       VARCHAR2(14) ,  
  loc         VARCHAR2(13) ,  
  create_date DATE DEFAULT SYSDATE  
);
```



Confirming and Displaying the Table Structure

- Use the DESCRIBE command to display the structure of a table including column name, datatype and nullability

```
DESC[RIBE] tablename
```



Confirming and Displaying the Table Structure

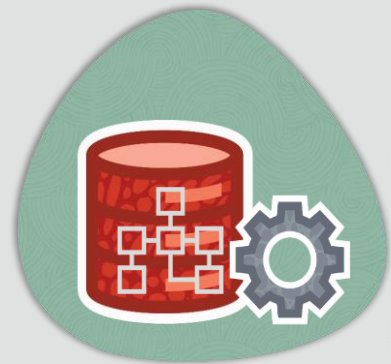
- Confirm table creation and columns:

```
DESCRIBE dept;
```

Table DEPT created.

Elapsed: 00:00:00.037

| Name | Null? | Type |
|-------------|-------|--------------|
| DEPTNO | | NUMBER(2) |
| DNAME | | VARCHAR2(14) |
| LOC | | VARCHAR2(13) |
| CREATE_DATE | | DATE |



ORACLE
Academy

ADS – Section 2
Database Basics – Part 2

Copyright © 2022, Oracle and/or its affiliates. Oracle, Java, and MySQL are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

24

Here we describe the table we just created called dept and we can see the basic characteristics of the table.

Table Columns Must Have a Data Type

| Data Type | Description |
|-----------------------|---|
| VARCHAR2(size) | Variable-length character data (A maximum size must be specified; minimum size is 1) |
| CHAR(size) | Fixed-length character data of length (size) bytes (Default and minimum size is 1; maximum size is 2,000) |
| NUMBER(p, s) | Variable-length numeric data. Precision is p, and scale is s (Precision is the total number of decimal digits, and scale is the number of digits to the right of the decimal point; precision can range from 1 to 38, and scale can range from -84 to 127) |
| DATE | Date and time values to the nearest second between January 1, 4712 B.C, and December 31, 9999 A.D |
| LONG | Variable-length character data (up to 2 GB) |

When we created columns, we give them data types. For characters, the most common one is called a VARCHAR(2), as compared to a CHAR.

For numbers you can specify the precision and scale; whole and decimal number of digits. Date datatype is used for storage and efficient manipulation of data information.

Additional Data Types

| Data Type | Description |
|-------------------|--|
| CLOB | A character large object (CLOB) containing single-byte or multibyte characters |
| NCLOB | A CLOB containing Unicode characters (Both fixed-width and variable-width character sets are supported, both using the database national character set) |
| RAW (Size) | Raw binary data of length size bytes |
| LONG RAW | Raw binary data of variable length up to 2 GB |
| BLOB | A binary large object. |
| BFILE | Binary data stored in an external file (up to 4 GB) |
| ROWID | Base 64 string representing the unique address of a row in its table (This data type is primarily for values returned by the ROWID pseudocolumn) |

These are some additional available data types that are used for more specific data management needs or application development. These data types are not often encountered. ROWID is the unique address of a row in a table. Some applications specifically use this value for performance, security, or other reasons to access a specific data location in the database. But this does not actually define any specific type of data.

Creating a Table with Different Data Types

- Here is an example of a create table statement that includes columns with the different data types
 - Note: Create table statements can be quite complex

```
CREATE TABLE print_media(  
  product_id NUMBER(6) ,  
  id         NUMBER(6) ,  
  description VARCHAR2(100) ,  
  composite  BLOB ,  
  msourcetext CLOB ,  
  finaltext  CLOB ,  
  photo      BLOB ,  
  graphic    BFILE  
);
```



Creating Tables

- Confirm table creation and columns:

```
DESCRIBE print_media;
```

Table PRINT_MEDIA created.

Elapsed: 00:00:00.041

| Name | Null? | Type |
|-------------|-------|-----------------|
| PRODUCT_ID | | NUMBER(6) |
| ID | | NUMBER(6) |
| DESCRIPTION | | VARCHAR2(100) |
| COMPOSITE | | BLOB |
| MSOURCETEXT | | CLOB |
| FINALTEXT | | CLOB |
| PHOTO | | BLOB |
| GRAPHIC | | BINARY FILE LOB |



ORACLE
Academy

ADS – Section 2
Database Basics – Part 2

Copyright © 2022, Oracle and/or its affiliates. Oracle, Java, and MySQL are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

28

Date Data Types



- Dates and times are very important in databases
- In many businesses, precise identification of when events occur is critical

| Data Type | Description |
|---------------------------------|---|
| TIMESTAMP | Enables storage of time as a date with fractional seconds (TIMESTAMP stores the year, month, day, hour, minute, the second value of the DATE data types, and the fractional seconds value. There are several variations of this data type, such as WITH TIMEZONE and WITH LOCALTIMEZONE.) |
| INTERVAL YEAR TO MONTH | Enables storage of time as an interval of years and months (Used to represent the difference between two datetime values in which the only significant portions are the year and month) |
| INTERVAL DAY TO SECOND | Enables storage of time as an interval of days, hours, minutes, and seconds (Used to represent the precise difference between two datetime values) |
| TIMESTAMP WITH TIME ZONE | Variant of TIMESTAMP that includes a time zone region name or time zone offset in its value |

Date Data Types

- When inserting data into a timestamp column, you need to specify the:
 - Calendar day, month, and year
 - Time in hours, minutes, and seconds
- Example of TIMESTAMP data type:

```
CREATE TABLE table_ts(  
    c_id NUMBER(6),  
    c_ts TIMESTAMP  
);
```

```
INSERT INTO table_ts  
VALUES(1, '01-JAN-2022 2:00:00');
```

```
SELECT * FROM table_ts;
```

ORACLE
Academy

ADS – Section 2
Database Basics – Part 2

Copyright © 2022, Oracle and/or its affiliates. Oracle, Java, and MySQL are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

30

DEFAULT Option

- Specify a default value for a column during CREATE TABLE
- This option prevents null values from entering the columns when a row is inserted without a value for the column

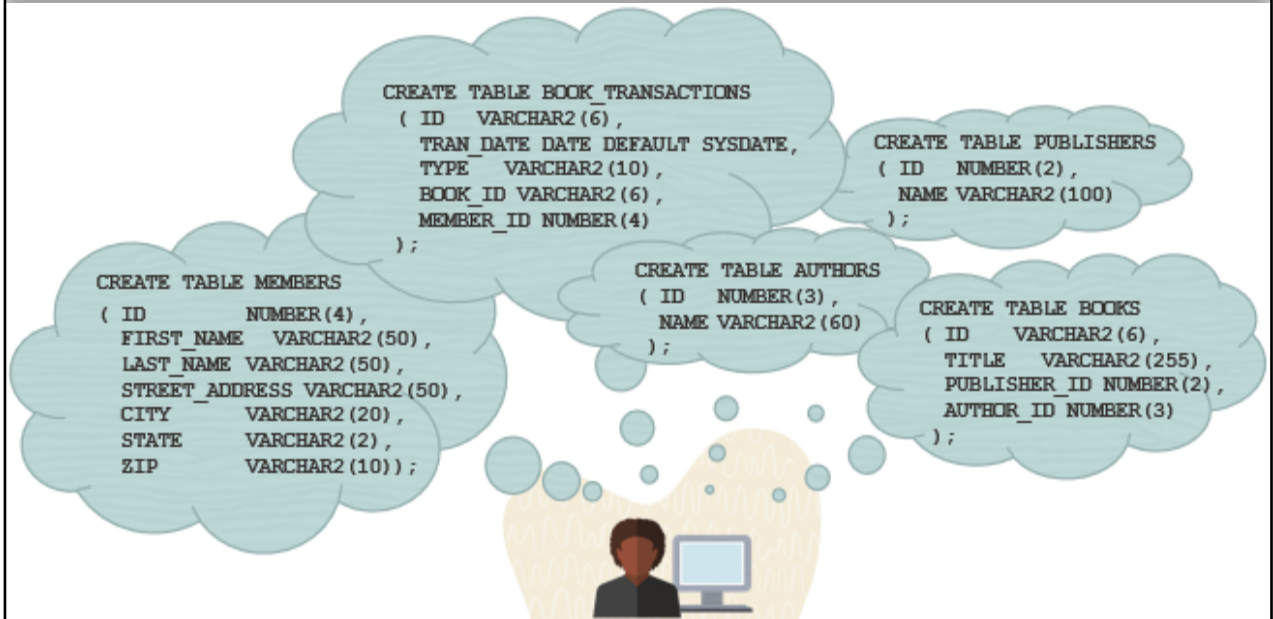
```
CREATE TABLE hire_dates(  
  id          NUMBER(8),  
  hire_date   DATE DEFAULT SYSDATE  
);
```

- Literal values, expressions, or SQL functions are legal values

```
INSERT INTO hire_dates values(45, NULL);  
INSERT INTO hire_dates(id) values(35);
```

In this case table, we created a column called hire_date with datatype DATE. We also created a default value in the column of current system date. This option prevents null values from entering the columns when a row is inserted without a value for the column.

Creating Tables for a Simple Library Application



Create a simple library application with all its associated tables.

Creating Tables for a Simple Library Application

- First, create the authors table which contains the author id and name
- Then create a members table with all the relevant information for the member joining the library

```
CREATE TABLE authors(  
  id          NUMBER(3) ,  
  name        VARCHAR2(60)  
);
```

```
CREATE TABLE members(  
  id          NUMBER(4) ,  
  first_name   VARCHAR2(50) ,  
  last_name    VARCHAR2(50) ,  
  street_address VARCHAR2(50) ,  
  city         VARCHAR2(20) ,  
  state        VARCHAR2(2) ,  
  zip          VARCHAR2(10)  
);
```

Creating Tables for a Simple Library Application

- The creation of the publishers and books tables is next
- These tables contain columns which cannot be null

```
CREATE TABLE publishers(  
  id          NUMBER(2) ,  
  name        VARCHAR2(100) NOT NULL  
);
```

```
CREATE TABLE books(  
  id          VARCHAR2(6) ,  
  title        VARCHAR2(255) NOT NULL ,  
  publisher_id NUMBER(2) ,  
  author_id    NUMBER(3)  
);
```

The publishers table contains the name column which cannot be null, so a value will have to be included anytime a row of data is added to the publishers table. Same applies for the title column in the books table.

Notice the use of the VARCHAR(2) datatype for the book title. Since we don't know how long a book title will be, we define it as a maximum of 255 characters, but can be any length from 0 to 255.

How to Change a Table After Its Created

- The ALTER TABLE command can be used to change something in a table
- For example, add an additional column of information, or adjust the number of characters needed in a column



ALTER TABLE Statement

Use the ALTER TABLE statement to change the table structure:

- Add a column
- Modify an existing column definition
- Define a default value for the new column
- Drop a column
- Rename a column
- Change a table to read-only status



ALTER TABLE Statement

- Use the ALTER TABLE statement to add, modify, or drop columns:

```
ALTER TABLE table
ADD          (column data type [DEFAULT expr]
              [, column data type]...);
```

```
ALTER TABLE table
MODIFY       (column data type [DEFAULT expr]
              [, column data type]...);
```

```
ALTER TABLE table
DROP (column [, column] ...);
```

Create a Sample Table Called Employees

- Create an employees table that we will use for the next set of examples
- Execute the describe statement to verify the creation

```
CREATE TABLE employees(  
    employee_id NUMBER(6) ,  
    last_name    VARCHAR(30) ,  
    first_name   VARCHAR(30) ,  
    deptno       NUMBER(6) ,  
    salary        NUMBER(7) ,  
    hire_date     DATE DEFAULT SYSDATE  
);
```

```
DESCRIBE employees;
```

Add a Column to the Table Called Termination_date

- You use the ADD clause to add columns:

```
ALTER TABLE employees  
ADD termination_date DATE;
```

- The new column becomes the last column:

| EMPLOYEE_ID | LAST_NAME | HIRE_DATE | TERMINATION_DATE |
|-------------|-----------|-------------|------------------|
| 100 | King | 17-Jun-1987 | - |
| 101 | Kochhar | 21-Sep-1989 | - |
| 102 | De Haan | 13-Jan-1993 | - |
| 200 | Whalen | 17-Sep-1987 | - |

```
DESCRIBE employees;
```

ORACLE
Academy

ADS – Section 2
Database Basics – Part 2

Copyright © 2022, Oracle and/or its affiliates. Oracle, Java, and MySQL are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

39

In this example, there's a need to add a column for the employee termination date. We can use the alter table command and add a column called termination_date with date datatype. Note: If a table already contains rows when a column is added, the new column is initially null or takes the default value for all rows. You can add a mandatory NOT NULL column to a table that contains data in the other columns only if you specify a default value. You can add a NOT NULL column to an empty table without the default value.

Modifying a Column

- You can change a column's data type, size, and default value:

```
ALTER TABLE employees  
MODIFY last_name VARCHAR2(35);
```

- A changed default value affects only subsequent insertions in the table
- The modifications are subject to certain conditions

```
DESCRIBE employees;
```

```
INSERT INTO employees (last_name)  
VALUES ('Name that is more than 35 characters');
```

The first example shows how to modify the last_name column to be able to fit up to 35 characters.

Therefore, if you try to insert more than 35 characters, the insert statement will fail.

Dropping a Column

- Use the DROP COLUMN clause to drop columns that you no longer need:

```
ALTER TABLE employees  
DROP (termination_date);
```

| EMPLOYEE_ID | LAST_NAME | HIRE_DATE |
|-------------|-----------|-------------|
| 100 | King | 17-Jun-1987 |
| 101 | Kochhar | 21-Sep-1989 |
| 102 | De Haan | 13-Jan-1993 |
| 200 | Whalen | 17-Sep-1987 |

```
DESCRIBE employees;
```

Drop column is part of the alter table command; you need to specify the column name you want to drop.

Here we are dropping the termination_date column we recently created. All data that was in the dropped column is removed from the table as well.

SET UNUSED Option

- The SET UNUSED option marks one or more columns as unused

```
ALTER TABLE    <table_name>  
SET      UNUSED(<column_name> [ , <column_name>]);
```

or

```
ALTER TABLE    <table_name>  
SET      UNUSED COLUMN <column_name> [ , <column_name>];
```

- You use the DROP UNUSED COLUMNS option to remove the columns that are marked as unused

```
ALTER TABLE <table_name>  
DROP  UNUSED COLUMNS;
```

In some cases, you may not want to drop a column you no longer use because you may want to keep the data around. In that case, you can set the column “UNUSED”; the column and data on it will no longer be visible or searchable but will remain in the database.

SET UNUSED Option

- Set the hire_date column from the employees table as UNUSED

```
ALTER TABLE employees  
SET UNUSED (hire_date);
```

- Use the DROP UNUSED COLUMNS to remove all columns that are currently marked as unused from the table

```
ALTER TABLE employees  
DROP UNUSED COLUMNS;
```

On setting a column as “UNUSED”, you have the option of dropping that column. You can use the DROP UNUSED COLUMNS statement when you want to reclaim the extra disk space from the unused columns in the table. If the table contains no unused columns, the statement returns with no errors.

Compressed Tables and Drop

- The Oracle Database allows data to be compressed to save storage and improve read operations
- Columns in compressed tables cannot be dropped unless uncompressed
- Steps:

1. Find out which tables are compressed

```
SELECT table_name, compression, compress_for  
FROM user_tables;
```

2. Uncompress the employees table

```
ALTER TABLE employees nocompress;
```

3. Drop the unused columns

```
ALTER TABLE employees  
DROP UNUSED COLUMNS;
```

To uncompress a table use the nocompress clause in the alter table. Once the table is no longer compressed then columns can be dropped.

Read-Only Tables

- You can use the ALTER TABLE syntax to:
 - Put a table in read-only mode, which prevents DDL or DML changes during table maintenance
 - Put the table back into read/write mode

```
ALTER TABLE employees READ ONLY;

-- perform table maintenance and then
-- return table back to read/write mode

ALTER TABLE employees READ WRITE;
```

Here are the guidelines for putting a table in read-only mode:

- You can specify READ ONLY to place a table in read-only mode
- When a table is in read-only mode, you cannot issue any DML statements that affect the table or any SELECT ... FOR UPDATE statements
- You can issue DDL statements as long as they do not modify any data in the table
- Operations on indexes associated with the table are allowed when the table is in read-only mode
- The Alter table READ/WRITE returns a read-only table to read/write mode

Note: You can drop a table that is in READ ONLY mode.

Dropping a Table

- Moves a table to the recycle bin
- Removes the table and its data if the PURGE clause is specified
- Invalidates dependent objects and removes object privileges on the table

```
DROP TABLE dept;
```

The DROP command is executed only in the data dictionary, so access to the table contents is not required. The space used by the table is not reclaimed until the tablespace is made read/write again, and then the required changes can be made to the block segment headers, and so on.

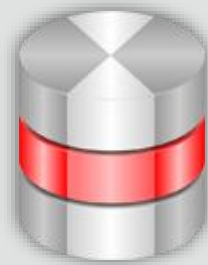
An abstract landscape illustration. The background is a light beige color. In the upper left, there are white, stylized mountain peaks. In the upper right, there is a bright yellow sun with rays. Below the sun, there are colorful, abstract shapes in shades of purple, red, orange, and black, resembling a stylized mountain or a group of people. In the lower left, there is a solid green hill. In the lower right, there is a green hill with a pattern of white, stylized lines. The title "Inserting and Updating Data" is centered in the upper half of the image.

Inserting and Updating Data

ORACLE
Academy

Data Manipulation Language

- A DML statement is executed when you:
 - Add new rows to a table (INSERT)
 - Modify existing rows in a table (UPDATE)
 - Remove existing rows from a table (DELETE)
- A database transaction consists of a collection of DML statements that form a logical unit of work



Inserting Data

- Let's start by inserting data into a table



```
INSERT INTO AUTHORS  
VALUES ('002', 'Oscar Wilde');  
INSERT INTO AUTHORS  
VALUES ('003', 'George Shaw');  
INSERT INTO AUTHORS  
VALUES ('004', 'Leo');
```

INSERT Statement Syntax

- Add rows to a table by using the INSERT statement:

```
INSERT INTO table [(column [, column...])]  
VALUES            (value [, value...]);
```

- In the syntax:
 - **Table** is the name of the table
 - **Column** is the name of the column in the table that you want to populate
 - **Value** is the corresponding value for the column
- With this syntax, only one row is inserted at a time

Inserting Rows

- If you insert a row that contains values for each column, the column list is not required in the INSERT clause
- List values in the default order of the columns in the table
- A value must be provided for each column

```
INSERT INTO employees  
VALUES (113, 'Louis', 'Popp', 6900, 76000, SYSDATE);
```

Inserting Rows

- Alternatively, list the columns in the INSERT clause

```
INSERT INTO employees (employee_id, last_name, first_name,  
deptno, salary, hire_date)  
VALUES (114, 'Jim', 'Drake', 6900, 76000, SYSDATE);
```

- List values in the same order as listed fields
- Enclose character and date values within single quotation marks

```
SELECT *  
FROM employees;
```

Another way to insert data into a table is to list the columns you want to insert data into. In this case, we specifically list which columns we are inserting data into and then we specify the value we are inserting into each one in the values clause.

The columns don't have to be in sequential definition order as in the previous example.

Inserting Rows with Null Values

- Explicit method: Omit the column from the column list

```
INSERT INTO employees (employee_id, last_name, first_name)
VALUES (115, 'Michael', 'Smith');
```

- Implicit method: Specify the NULL keyword in the VALUES clause

```
INSERT INTO employees
VALUES (116, 'Jan', 'Lazar', NULL, NULL, NULL);
```

- See what's in the table

```
SELECT *
FROM employees;
```

Be sure that you can use null values in the targeted column by verifying the Null status with the DESCRIBE command.

The Oracle server automatically enforces all data types, data ranges, and data integrity constraints. Any column that is not listed explicitly obtains a null value in the new row unless there are default values for the missing columns that are used.

Common errors that can occur during user input are checked in the following order:

- A mandatory value is missing for a NOT NULL column
- A duplicate value violates any unique or primary key constraint
- The Any value violates a CHECK constraint
- A foreign key violates the referential integrity constraint
- Data type mismatches or values are too wide to fit in the column
- Use of the column list is recommended because it makes the INSERT statement more readable and reliable and less prone to mistakes

Updating Data

- Updating data is a very common operation in databases because data can change



Changing Data in a Table (Update)

EMPLOYEES

| EMPLOYEE_ID | FIRST_NAME | LAST_NAME | EMAIL | PHONE_NUMBER | HIRE_DATE | JOB_ID | SALARY | COMMISSION_PCT | MANAGER_ID | DEPARTMENT_ID |
|-------------|------------|-----------|----------|--------------|-------------|----------|--------|----------------|------------|---------------|
| 100 | Steven | King | SKING | 515.123.4567 | 17-Jun-1987 | AD_PRE S | 24000 | - | - | 6900 |
| 101 | Neena | Kochhar | NKOCHHAR | 515.123.4568 | 21-Sep-1989 | AD_VP | 17000 | - | 100 | 6900 |
| 102 | Lex | De Haan | LDEHAAN | 515.123.4569 | 13-Jan-1993 | AD_VP | 17000 | - | 100 | 6900 |
| 200 | Jennifer | Whalen | JWHALEN | 515.123.4444 | 17-Sep-1987 | AD_ASST | 4400 | - | 101 | 10 |

...

Update rows in the EMPLOYEES table:

| EMPLOYEE_ID | FIRST_NAME | LAST_NAME | EMAIL | PHONE_NUMBER | HIRE_DATE | JOB_ID | SALARY | COMMISSION_PCT | MANAGER_ID | DEPARTMENT_ID |
|-------------|------------|-----------|----------|--------------|-------------|----------|--------|----------------|------------|---------------|
| 100 | Steven | King | SKING | 515.123.4567 | 17-Jun-1987 | AD_PRE S | 24000 | - | - | 7000 |
| 101 | Neena | Kochhar | NKOCHHAR | 515.123.4568 | 21-Sep-1989 | AD_VP | 17000 | - | 100 | 7000 |
| 102 | Lex | De Haan | LDEHAAN | 515.123.4569 | 13-Jan-1993 | AD_VP | 17000 | - | 100 | 7000 |
| 200 | Jennifer | Whalen | JWHALEN | 515.123.4444 | 17-Sep-1987 | AD_ASST | 4400 | - | 101 | 10 |

ORACLE
Academy

ADS – Section 2
Database Basics – Part 2

Copyright © 2022, Oracle and/or its affiliates. Oracle, Java, and MySQL are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

55

Let's consider our employees table where we have employees in department_id 6900. Maybe because of internal reorganization, these employees will be moving to a new department whose id is 7000.

UPDATE Statement Syntax

- Modify existing values in a table with the UPDATE statement:

```
UPDATE table ←  
SET      column = value [, column = value, ...]  
[WHERE condition];
```

–Note: It is recommended that the UPDATE statement be on a line of its own

- In general, use the primary key column in the WHERE clause to identify a single row for update
- Update more than one row at a time (if required)

The update statement modifies existing values in a table.

In the syntax:

- table is the name of the table
- column is the name of the column in the table to populate
- value is the corresponding value or subquery for the column
- condition identifies the rows to be updated and consists of column names, expressions, constants, subqueries, and comparison operators

Updating Rows in a Table

- Update department_id from 6900 to 7000 for every employee in the table

```
UPDATE employees  
SET    deptno = 7000  
WHERE  deptno = 6900;
```

- Verify the update

```
SELECT *  
FROM employees;
```

The easiest way to do a global table update is to use the WHERE clause, where we specify a condition that must be met for a particular column. In this case, we are setting the deptno to 7000 for every row that has the current deptno of 6900.

Updating Rows in a Table

- Values for a specific row or rows are modified if you specify the WHERE clause:

```
UPDATE employees  
SET    deptno = 50  
WHERE  employee_id = 113;
```

- Values for all the rows in the table are modified if you omit the WHERE clause:

```
UPDATE employees  
SET    deptno = 110;
```

If we want to update specific records (rows) in the database, the WHERE clause can be used to specify which records to update. On the other hand, if you want to update every record in a table, omit the WHERE clause and every row will change. In the bottom example we are setting every employees deptno to 110.

Updating Rows in a Table

- Specify SET column_name = NULL to update a column value to NULL

```
UPDATE employees  
SET    deptno = NULL  
WHERE  employee_id = 113;
```

- NULL is an important value in databases because it fills a void where a particular value is not known

If a column allows NULL values, a record can be set to null. In the example above, employee 113 may have left the company and so we are setting the depot to NULL since there is no longer a valid deptno for that employee.

Is NULL a VALUE?

- A null value in a relational database is **used when the value in a column is unknown or missing**
- A null is neither an empty string (for character or datetime data types) nor a zero value (for numeric data types)

Deleting Data

- You can remove existing rows from a table by using the DELETE statement



```
DELETE FROM employees  
WHERE last_name = 'Louis';
```

Removing Data (Row) From a Table

EMPLOYEES

| EMPLOYEE_ID | FIRST_NAME | LAST_NAME | EMAIL | PHONE_NUMBER | HIRE_DATE | JOB_ID | SALARY | COMMISSION_PCT | MANAGER_ID | DEPARTMENT_ID |
|-------------|------------|-----------|----------|--------------|-------------|----------|--------|----------------|------------|---------------|
| 100 | Steven | King | SKING | 515.123.4567 | 17-Jun-1987 | AD_PRE S | 24000 | - | - | 6900 |
| 101 | Neena | Kochhar | NKOCHHAR | 515.123.4568 | 21-Sep-1989 | AD_VP | 17000 | - | 100 | 6900 |
| 102 | Lex | De Haan | LDEHAAN | 515.123.4569 | 13-Jan-1993 | AD_VP | 17000 | - | 100 | 6900 |
| 200 | Jennifer | Whalen | JWHALEN | 515.123.4444 | 17-Sep-1987 | AD_ASST | 4400 | - | 101 | 10 |

...

Last row deleted from the **EMPLOYEES** table:

| EMPLOYEE_ID | FIRST_NAME | LAST_NAME | EMAIL | PHONE_NUMBER | HIRE_DATE | JOB_ID | SALARY | COMMISSION_PCT | MANAGER_ID | DEPARTMENT_ID |
|-------------|------------|-----------|----------|--------------|-------------|----------|--------|----------------|------------|---------------|
| 100 | Steven | King | SKING | 515.123.4567 | 17-Jun-1987 | AD_PRE S | 24000 | - | - | 7000 |
| 101 | Neena | Kochhar | NKOCHHAR | 515.123.4568 | 21-Sep-1989 | AD_VP | 17000 | - | 100 | 7000 |
| 102 | Lex | De Haan | LDEHAAN | 515.123.4569 | 13-Jan-1993 | AD_VP | 17000 | - | 100 | 7000 |

ORACLE
Academy

ADS – Section 2
Database Basics – Part 2

Copyright © 2022, Oracle and/or its affiliates. Oracle, Java, and MySQL are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

62

In our employees table, we want to delete the employee Jennifer Whalen from the table.

DELETE Statement

- You can remove existing rows from a table by using the DELETE statement:

```
DELETE [FROM] table  
[WHERE condition];
```

- You can delete rows based on specific conditions or delete all rows in the table

Deleting Rows From a Table

- Specific rows are deleted if you include the WHERE clause:

```
DELETE FROM employees  
WHERE last_name = 'Whalen';
```

- All rows in the table are deleted if you omit the WHERE clause:

```
DELETE FROM employees;
```

Here is an example of removing rows identified in the WHERE clause. In this case, we are deleting the employee with the name Whalen, in the first example. In the second example all rows in the table employees are deleted.

Violating Constraints

- You cannot delete a row that contains a primary key which is used as a foreign key in another table
- In this example, an attempt to delete department 60 from the DEPARTMENTS table results in an error because that department number is used as a foreign key in the EMPLOYEES table

```
DELETE FROM departments
WHERE      department_id = 60;
```

```
Error starting at line 1 in command:
DELETE FROM departments
WHERE department_id = 60
Error report:
SQL Error: ORA-02292: integrity constraint (ORA1.JHIST_DEPT_FK) violated - child record found
02292. 00000 - "integrity constraint (%s.%s) violated - child record found"
*Cause:      attempted to delete a parent key value that had a foreign
              dependency.
*Action:     delete dependencies first then parent or disable constraint.
```

ORACLE
Academy

ADS – Section 2
Database Basics – Part 2

Copyright © 2022, Oracle and/or its affiliates. Oracle, Java, and MySQL are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

65

If you attempt to delete a record with a value that is tied to an integrity constraint, an error is returned.

If the parent record that you attempt to delete has child records, you receive the "child record found" violation.

TRUNCATE Statement – Empty a table

The TRUNCATE Statement:

- Removes all rows from a table, leaving the table empty and the table structure intact
- Is a DDL statement rather than a DML statement; cannot easily be undone
- Syntax:

```
TRUNCATE TABLE table_name;
```

- Example:


```
TRUNCATE TABLE employees;
```

The TRUNCATE statement is a more efficient method to remove all rows from a table or cluster.

Removing rows with the TRUNCATE statement is faster than removing them with the DELETE statement for the following reasons:

- The TRUNCATE statement is a DDL statement and generates no rollback information, which limits recovery of table. Rollback information is covered later in this lesson.
- Truncating a table does not fire the delete triggers of the table.

If the table is the parent of a referential integrity constraint, you cannot truncate the table. You need to disable the constraint before issuing the TRUNCATE statement.

An abstract landscape illustration. The top half features a pale yellow sky with a bright yellow sun in the upper right corner. Below the sky, there are stylized mountains: one on the left is white with fine white lines, and a larger one on the right is composed of various colored patches (purple, red, orange, black, green) with white stitching-like lines. The bottom half of the image shows a green foreground. The left side is a solid green slope, while the right side is a green area filled with a dense, repeating geometric pattern of small triangles and lines.

Retrieving Data

ORACLE
Academy

Retrieving Data

```
SELECT id, name  
FROM authors
```

| ID | NAME |
|-----|---------------------|
| 200 | P.G. Wodehouse |
| 300 | George Bernard Shaw |
| 100 | Leo Tolstoy |

I would like to retrieve the data from the **AUTHORS** and **BOOKS** tables. Is that possible?

```
SELECT *  
FROM books
```

Here is the information

| ID | TITLE | PUBLISHER_ID | AUTHOR_ID |
|----|--------------------------|--------------|-----------|
| 3 | An Unsocial Socialist | 30 | 300 |
| 1 | War and Peace | 10 | 100 |
| 2 | The Clicking of Cuthbert | 20 | 200 |

ORACLE
Academy

ADS – Section 2
Database Basics – Part 2

Copyright © 2022, Oracle and/or its affiliates. Oracle, Java, and MySQL are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

68

The select statement is used for data retrieval. The easiest way to retrieve every piece of data in a table is by using the “star” operative in the select statement, which effectively specifies no restrictions on the data being queried.

Basic SELECT Statement

- **SELECT** identifies the columns to be displayed
- **FROM** identifies the table that contains those columns

```
SELECT {*[DISTINCT] column|expression [alias],...}  
FROM    table;
```

- In the syntax:
 - **SELECT *** selects all columns
 - **DISTINCT** suppresses duplicates
 - **column|expression** selects the named column or the expression
 - **alias** gives different headings to the selected columns

Note: Throughout this lesson, the words keyword, clause, and statement are used as follows:

- A keyword refers to an individual SQL element; for example, **SELECT** and **FROM** are keywords
- A clause is a part of a SQL statement; for example, **SELECT employee_id, last_name**
- A statement is a combination of two or more clauses; for example, **SELECT * FROM employees**

Selecting All Columns

- All columns of a table can be displayed by placing an * after keyword SELECT

```
SELECT *  
FROM departments;
```

| DEPARTMENT_ID | DEPARTMENT_NAME | MANAGER_ID | LOCATION_ID |
|---------------|-----------------|------------|-------------|
| 10 | Administration | 200 | 1700 |
| 20 | Marketing | 201 | 1800 |
| 50 | Shipping | 124 | 1500 |
| 60 | IT | 103 | 1400 |
| 80 | Sales | 149 | 2500 |
| 90 | Executive | 100 | 1700 |
| 110 | Accounting | 205 | 1700 |

Here we are selecting all rows and all columns from the departments table.

Selecting Specific Columns

- You can use the SELECT statement to display specific columns of the table by indicating the column names in the order you would like to see them, separated by commas

```
SELECT department_id, location_id  
FROM departments;
```

| DEPARTMENT_ID | LOCATION_ID |
|---------------|-------------|
| 10 | 1700 |
| 20 | 1800 |
| 50 | 1500 |
| 60 | 1400 |
| 80 | 2500 |

Arithmetic Expressions

- Create expressions with number and date data by using arithmetic operators
- Column names, numeric constants and arithmetic operators can be used in an arithmetic expression
- Arithmetic operators can be used in any clause of a SQL statement except FROM

| Operator | Description |
|----------|-------------|
| + | Add |
| - | Subtract |
| * | Multiply |
| / | Divide |

Arithmetic operators can be used as part of SQL statements including selects. The most common operators are plus, minus, multiply, and divide.

Note: With the DATE and TIMESTAMP data types, you can only use the addition and subtraction operators.

Using Arithmetic Operators

- The addition operator is used to calculate a salary increase of \$300 for all employees
- **SALARY + 300** is displayed as the column heading

```
SELECT last_name, salary, salary + 300
FROM   employees;
```

| LAST_NAME | SALARY | SALARY+300 |
|-----------|--------|------------|
| King | 24000 | 24300 |
| Kochhar | 17000 | 17300 |
| De Haan | 17000 | 17300 |
| Whalen | 4400 | 4700 |

When using an arithmetic operator as part of a select statement, in this case the salary column + 300, the resultant calculated column, salary + 300, is not a new column in the EMPLOYEES table; it is for display only. By default, the name of a new column comes from the calculation that generated it (in this case, salary + 300).

Operator Precedence

- The rules of Precedence for operators are:
 - Multiplication and division are evaluated before addition and subtraction
 - Operators of the same priority are evaluated from left to right
 - Parentheses are used to override the default precedence or to clarify the statement

```
SELECT last_name, salary, 12*salary+100
FROM employees;
```

| LAST_NAME | SALARY | 12*SALARY+100 |
|-----------|--------|---------------|
| King | 24000 | 288100 |
| Kochhar | 17000 | 204100 |
| De Haan | 17000 | 204100 |
| Whalen | 4400 | 52900 |
| Higgins | 12000 | 144100 |
| Gietz | 8300 | 99700 |

...

In this example, salary is multiplied by 12 and that results get 100 added to it.

Operator Precedence

- You can override the rules of precedence by using parentheses to specify the order in which the operators are to be executed

```
SELECT last_name, salary, 12*(salary+100)
FROM employees;
```

| LAST_NAME | SALARY | 12*(SALARY+100) |
|-----------|--------|-----------------|
| King | 24000 | 289200 |
| Kochhar | 17000 | 205200 |
| De Haan | 17000 | 205200 |
| Whalen | 4400 | 54000 |
| Higgins | 12000 | 145200 |
| Gietz | 8300 | 100800 |

ORACLE
Academy

...

ADS – Section 2
Database Basics – Part 2

Copyright © 2022, Oracle and/or its affiliates. Oracle, Java, and MySQL are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

75

In this case, 100 will be added to salary before that results gets multiplied by 12.

Null Values in Arithmetic Expressions

- Any arithmetic expression containing a null value will evaluate to null

```
SELECT last_name, 12*salary*commission_pct
FROM employees;
```

| LAST_NAME | 12*SALARY*COMMISSION_PCT |
|-----------|--------------------------|
| King | - |
| Gietz | - |
| Zlotkey | 25200 |
| Abel | 39600 |
| Taylor | 20640 |
| Grant | 12600 |
| Mourgos | - |

...

Defining a Column Alias

- A column alias renames a column heading
- A column alias:
 - Is useful with calculations
 - Immediately follows the column name
 - There can also be the optional AS keyword between the column name and the alias
 - Requires double quotation marks if it contains spaces or special characters or if it is case-sensitive, the default is all uppercase

Developers will many times use column aliases to improve readability or to make operations easier.

Using Column Aliases

- Keyword AS is optional - Column names appear uppercase by default

```
SELECT last_name AS name,  
       commission_pct comm  
FROM employees;
```

| NAME | COMM |
|---------|------|
| King | - |
| Kochhar | - |
| Whalen | - |
| Higgins | - |

...

- Column names enclosed in quotes will appear as entered

```
SELECT last_name "Name",  
       salary*12 "Annual Salary"  
FROM employees;
```

| NAME | Annual Salary |
|---------|---------------|
| King | 288000 |
| Kochhar | 204000 |
| Whalen | 204000 |
| Higgins | 52800 |

...

In these examples, we are creating an alias for the last_name column simply as name. The result is displayed the column name in the result will be name and not last name. To get the literal value displayed, put the alias name between quotations, otherwise they get all capped or may get evaluated as an expression.

Concatenation Operator

- Links columns or character strings to other columns
- Is represented by two vertical bars (||)
- Creates a column that is a character expression
- Concatenating a NULL with a character results in a character string

```
SELECT last_name || job_id AS "Employees"  
FROM employees;
```

| Employees |
|---------------|
| KingAD_PRES |
| KochharAD_VP |
| De HaanAD_VP |
| WhalenAD_ASST |

Sometimes it is useful to concatenate together multiple columns. The concatenation operator is represented by two vertical bars. When two columns are concatenated, they will be display back as one column with no spaces in between.

Literal Character Strings

- A literal is a character, a number, or a date that is included in the SELECT statement
- Date and character literal values must be enclosed within single quotation marks
- Each character string is output once for each row returned



The exact content of the literal string is displayed back in the result set.

Using Literal Character Strings

- Last_name and job_id for each employee is concatenated with a literal to give the returned rows more meaning

```
SELECT last_name || ' is a ' || job_id AS "Employee Details"  
FROM   employees;
```

| Employees |
|-----------------------|
| King is a AD_PRES |
| Kochhar is a AD_VP |
| De Haan is a AD_VP |
| Whalen is a AD_ASST |
| Higgins is a AC_MGR |
| Gietz is a AC_ACCOUNT |

...

The result is more readable than simply returning the last name and job id columns.

Duplicate Rows

- The default display of queries is all rows, including duplicate rows

```
SELECT department_id  
FROM employees;
```

| DEPARTMENT_ID |
|---------------|
| 90 |
| 90 |
| 90 |
| 10 |
| 110 |
| 110 |
| 80 |
| 80 |
| 80 |

...

Duplicate Rows

- To eliminate duplicate rows in the result, include the **DISTINCT** keyword in the **SELECT** clause immediately after the **SELECT** keyword

```
SELECT DISTINCT department_id  
FROM   employees;
```

| DEPARTMENT_ID |
|---------------|
| - |
| 90 |
| 10 |
| 110 |
| 80 |
| 50 |

...

Unlike the example in the last slide each department id is displayed only once in this result set.



Conditionally Retrieving Data

ORACLE
Academy

Order of Execution

The order of execution of a SELECT statement is as follows:

- FROM clause:
 - Locates the table that contains the data
- WHERE clause:
 - Restricts the rows to be returned
- SELECT clause:
 - Selects from the reduced data set the columns requested
- ORDER BY clause:
 - Orders the result set



Limiting the Rows That Are Selected

- Restrict the rows that are returned by using the WHERE clause:

```
SELECT *|{[DISTINCT] column|expression [alias],...}  
FROM   table  
[WHERE logical expression(s)];
```

- If the logical expression evaluates to true, the row meeting the condition is returned
- The WHERE clause follows the FROM clause

As we have already seen several times, the WHERE clause restricts the rows that are returned in a select statement. The where statement defines a logical condition that is evaluated to determine if the row will be returned or not.

Using the WHERE Clause

- Retrieve all employees in department 90

```
SELECT employee_id, last_name, job_id, department_id
FROM   employees
WHERE  department_id = 90;
```

| EMPLOYEE_ID | LAST_NAME | JOB_ID | DEPARTMENT_ID |
|-------------|-----------|---------|---------------|
| 100 | King | AD_PRES | 90 |
| 101 | Kochhar | AD_VP | 90 |
| 102 | De Haan | AD_VP | 90 |

In this example we specify in the where clause that we only want rows that contain department id 90.

Character Strings and Dates

- Character strings and date values are enclosed in single quotation marks
- Character values are case-sensitive, and date values are format-sensitive

```
SELECT last_name, job_id, department_id
FROM   employees
WHERE  last_name = 'Whalen';
```

- The default date display format is DD-Mon-YYYY

```
SELECT last_name
FROM   employees
WHERE  hire_date = '29-Jan-2000';
```

For the date data type, the default format is two digit day, three character month, and four digit year.

Comparison Operators

- There are several comparison operators included with SQL that help evaluate logical conditions

| Operator | Meaning |
|------------------|--------------------------------|
| = | Equal to |
| > | Greater than |
| >= | Greater than or equal to |
| < | Less than |
| <= | Less than or equal to |
| <> | Not equal to |
| BETWEEN...AND... | Between two values (inclusive) |
| IN (set) | Match any of a list of values |
| LIKE | Match a character pattern |
| IS NULL | Is a null value |

Please notice the last 2 listed here. The LIKE operator is used to match a character pattern in a column. The IS NULL test to see if a column value is set to NULL (as opposed to no value assigned).

Using Comparison Operators

- Retrieve records from the EMPLOYEES table where the salary is less than or equal to \$3,000

```
SELECT last_name, salary
FROM employees
WHERE salary <= 3000;
```

| LAST_NAME | SALARY |
|-----------|--------|
| Matos | 2600 |
| Vargas | 2500 |

- Where hire data is this year

```
SELECT last_name, salary
FROM employees
WHERE hire_date > '01-JAN-2022';
```

Here is an example of Retrieving records from the EMPLOYEES table where the salary is less than or equal to \$3,000

The example in the bottom looks for all the employees that were hired after Jan 1, 2022.

Range Conditions: BETWEEN Operator

- Use the BETWEEN operator to display rows based on a range of values:

```
SELECT last_name, salary
FROM employees
WHERE salary BETWEEN 2500 AND 3500 ;
```

Lower limit (points to 2500)
Upper limit (points to 3500)

| LAST_NAME | SALARY |
|-----------|--------|
| Rajs | 3500 |
| Davies | 3100 |
| Matos | 2600 |
| Vargas | 2500 |

–Note: When using BETWEEN, the lower value must be specified first

Membership Conditions: IN Operator

- Use the IN operator to test for values in a list:

```
SELECT employee_id, last_name, salary, manager_id
FROM   employees
WHERE  manager_id IN (100, 101, 201) ;
```

| EMPLOYEE_ID | LAST_NAME | SALARY | MANAGER_ID |
|-------------|-----------|--------|------------|
| 101 | Kochhar | 17000 | 100 |
| 102 | De Haan | 17000 | 100 |
| 149 | Zlotkey | 10500 | 100 |
| 124 | Mourgos | 5800 | 100 |
| 201 | Hartstein | 13000 | 100 |

–Note: Items in list can be in any order

If you want to specify a set of specific values to test for, use the IN Operators. Specify the values in between parenthesis and separated by commas.

In this example employees that have managers with ids of 100, 101, or 201 will be displayed.

Membership Conditions: NOT IN Operator

- Use the NOT IN operator to test for values not in a list:

```
SELECT employee_id, last_name, salary, manager_id
FROM   employees
WHERE  manager_id IN NOT IN (60, 90, 100) ;
```

| EMPLOYEE_ID | LAST_NAME | SALARY | MANAGER_ID |
|-------------|-----------|--------|------------|
| 200 | Whalen | 4400 | 101 |
| 205 | Higgins | 12000 | 101 |
| 206 | Gietz | 8300 | 205 |
| 149 | Zlotkey | 10500 | 100 |
| 174 | Abel | 11000 | 149 |
| 176 | Taylor | 8600 | 149 |

Conversely, if you specifically want some values excluded from a search, then use the NOT IN Operator. In this case employees whose managers DON'T have id's of 60, 90, or 100 will be returned.

Pattern Matching: LIKE Operator

- Use the LIKE operator to perform wildcard searches of valid search string values
- Search conditions can contain literal characters or numbers:
 - % denotes zero or more characters
 - _ denotes one character

```
SELECT first_name  
FROM   employees  
WHERE  first_name LIKE 'S%' ;
```

| FIRST_NAME |
|------------|
| Shelley |
| Steven |

In this case all employees with first names that start with an S will be returned.

Combining Wildcard Characters

- You can combine the two wildcard characters (% , _) with literal characters for pattern matching:

```
SELECT last_name  
FROM   employees  
WHERE  last_name LIKE '_o%' ;
```

| LAST_NAME |
|-----------|
| Kochhar |
| Lorentz |
| Mourgos |

In this case all employees with a last name that has an “o” as the second character of their last name will be returned.

Combining Wildcard Characters

- You can use the ESCAPE identifier to search for the actual % and _ symbols

```
SELECT employee_id, last_name, job_id
FROM employees
WHERE job_id LIKE '%SA\_%' ESCAPE '\';
```

- This will return records with SA_ in their job_id

| EMPLOYEE_ID | LAST_NAME | SALARY |
|-------------|-----------|--------|
| 149 | Zlotkey | SA_MAN |
| 174 | Abel | SA_REP |
| 176 | Taylor | SA_REP |
| 178 | Grant | SA_REP |

The ESCAPE identifier identifies the backslash (\) as the escape character. In the SQL statement, the escape character precedes the underscore (_) and causes the Oracle server to interpret the underscore literally.

Using the NULL Conditions

- Test for nulls with the IS NULL or IS NOT NULL operators:

```
SELECT last_name, manager_id
FROM   employees
WHERE  manager_id IS NULL;
```

| LAST_NAME | MANAGER_ID |
|-----------|------------|
| King | - |

- You cannot test with = because a null cannot be equal or unequal to any value

Defining Conditions Using the Logical Operators

- A logical condition combines the result of two component conditions to produce a single result based on those conditions
- Or, if using NOT, it inverts the result of a single condition

| Operator | Meaning |
|------------|--|
| AND | Returns TRUE if both component conditions are TRUE |
| OR | Returns TRUE if either component condition is TRUE |
| NOT | Returns TRUE if the condition is FALSE Returns FALSE if the condition is TRUE |

Using the AND Operator

- AND requires both component conditions to be true:

```
SELECT employee_id, last_name, job_id, salary
FROM   employees
WHERE  salary >= 10000
AND    job_id LIKE '%MAN%' ;
```

| EMPLOYEE_ID | LAST_NAME | JOB_ID | SALARY |
|-------------|-----------|--------|--------|
| 149 | Zlotkey | SA_MAN | 10500 |
| 201 | Hartstein | MK_MAN | 13000 |

- Note: All character searches are case sensitive and must be enclosed in quotation marks

In this example only salaries above 10000 and a job id that contains the letters “MAN” will be returned.

Using the OR Operator

- OR requires either component condition to be true

```
SELECT employee_id, last_name, job_id, salary
FROM   employees
WHERE  salary >= 10000
OR     job_id LIKE '%MAN%' ;
```

| EMPLOYEE_ID | LAST_NAME | JOB_ID | SALARY |
|-------------|-----------|---------|--------|
| 100 | King | AD_PRES | 24000 |
| 101 | Kochhar | AD_VP | 17000 |
| 102 | De Haan | AD_VP | 17000 |
| 205 | Higgins | AC_MGR | 12000 |
| 149 | Zlotkey | SA_MAN | 10500 |

In this case either a salary over 10000 or a job id containing “MAN” will be returned.

Using the NOT Operator

- NOT reverses the value of the condition

```
SELECT last_name, job_id
FROM employees
WHERE job_id NOT IN ('IT_PROG', 'ST_CLERK', 'SA_REP') ;
```

| LAST_NAME | JOB_ID |
|-----------|---------|
| King | AD_PRES |
| Kochhar | AD_VP |
| De Haan | AD_VP |
| Whalen | AD_ASST |
| Higgins | AC_MGR |

In this case all employees except those in IT_PROG, ST_CLERK, or SA_REP will be returned

Rules of Precedence

- Here is the list of operators in order of evaluation precedence
- Use parentheses to override rules of precedence

| Precedence | Operator |
|------------|-------------------------------|
| 1 | Arithmetic operators |
| 2 | Concatenation operator |
| 3 | Comparison conditions |
| 4 | IS [NOT] NULL, LIKE, [NOT] IN |
| 5 | [NOT] BETWEEN |
| 6 | Not equal to |
| 7 | NOT logical operator |
| 8 | AND logical operator |
| 9 | OR logical operator |

What happens when you have several of them in the same statement? Then the rules of precedence applies.

Rules of Precedence

- There are two conditions in this example for precedence of the AND operator
 - The first condition is that the job ID is AD_PRES, and the salary is greater than \$15,000
 - The second condition is that the job ID is SA_REP

```
SELECT last_name, job_id, salary
FROM employees
WHERE job_id = 'SA_REP'
OR job_id = 'AD_PRES'
AND salary > 15000;
```

Precedence of the AND Operator

| LAST_NAME | JOB_ID | SALARY |
|-----------|---------|--------|
| King | AD_PRES | 24000 |
| Abel | SA_REP | 11000 |
| Taylor | SA_REP | 8600 |
| Grant | SA_REP | 7000 |

ORACLE
Academy

ADS – Section 2
Database Basics – Part 2

Copyright © 2022, Oracle and/or its affiliates. Oracle, Java, and MySQL are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

103

Therefore, the SELECT statement reads as "Select the row if an employee is a president and earns more than \$15,000, or if the employee is a sales representative."

Rules of Precedence

- There are two conditions in this example for parentheses
 - The first condition is that the job ID is AD_PRES or SA_REP
 - The second condition is that the salary is greater than \$15,000

```
SELECT last_name, job_id, salary
FROM employees
WHERE (job_id = 'SA_REP'
OR job_id = 'AD_PRES')
AND salary > 15000;
```

Parentheses

| LAST_NAME | JOB_ID | SALARY |
|-----------|---------|--------|
| King | AD_PRES | 24000 |

ORACLE
Academy

ADS – Section 2
Database Basics – Part 2

Copyright © 2022, Oracle and/or its affiliates. Oracle, Java, and MySQL are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

104

Therefore, the SELECT statement reads as "Select the row if an employee is a president or a sales representative, and if the employee earns more than \$15,000."

An abstract landscape illustration. The top half features a pale yellow sky with a bright yellow sun in the upper right corner. Several white, wispy clouds are scattered across the sky. In the upper right, there is a large, colorful, abstract shape resembling a mountain or a cloud, composed of various colored segments (purple, red, orange, black, green, blue) with some internal line patterns. The bottom half of the image shows a green landscape. On the left, there is a solid green hill. On the right, there is a larger hill covered in a dense, intricate pattern of green lines forming a geometric, woven texture.

Ordering Data

ORACLE
Academy

ORDER BY Clause

- Numeric values are displayed lowest to highest
- Date values are displayed with the earliest value first
- Character values are displayed in alphabetical order
- Null values are displayed last in ascending order and first in descending order
- NULLS FIRST specifies that NULL values should be returned before non-NULL values
- NULLS LAST specifies that NULL values should be returned after non-NULL values

Using the ORDER BY Clause

- Sort the retrieved rows with the ORDER BY clause:
 - ASC: Ascending order (default)
 - DESC: Descending order
- The ORDER BY clause comes last in the SELECT statement:

```
SELECT    last_name, job_id, department_id, hire_date
FROM      employees
ORDER BY  hire_date ;
```

If the ORDER BY clause is not used, the sort order is undefined, and the Oracle server may not fetch rows in the same order for the same query twice. Use the ORDER BY clause to display the rows in a specific order.

Use the NULLS FIRST or NULLS LAST keywords to specify whether returned rows containing null values should appear first or last in the ordering sequence.

Sorting

- Sorting in descending order

```
SELECT    last_name, job_id, department_id, hire_date
FROM      employees
ORDER BY  hire_date DESC ;
```

- Sorting by column alias

```
SELECT    employee_id, last_name, salary*12 annsal
FROM      employees
ORDER BY  annsal ;
```

The default sort order is ascending:

- Numeric values are displayed with the lowest values first (for example, 1 to 999)
- Date values are displayed with the earliest value first (for example, 01-Jan-1992 before 01-Jan-1995)
- Character values are displayed in alphabetical order (for example, "A" first and "Z" last)
- By default, null values are displayed last for ascending sequences and first for descending sequences
 - You can change this by adding a NULLS FIRST or NULLS LAST option
- You can also sort by a column that is not in the SELECT list

Sorting

- Sorting by using the column's numeric position

```
SELECT    last_name, job_id, department_id, hire_date
FROM      employees
ORDER BY 3;
```

- Sorting by multiple columns

```
SELECT    last_name, department_id, salary
FROM      employees
ORDER BY department_id, salary DESC;
```

Substitution Variables

- When running a report dynamically restrict the data that is returned
- With substitution variables, create reports that prompt users to supply their own values to restrict the range of returned data




Substitution variables can be used to prompt a user to input a value that will be used as the condition for selection.

Using a Substitution Variable

- Use substitution variables to prompt for values
- Use a variable prefixed with a colon(:) to prompt the user for a value:

```
SELECT employee_id, last_name, salary, department_id
FROM   employees
WHERE  employee_id = :employee_num ;
```

An abstract landscape illustration. The top half features a pale yellow sky with a bright yellow sun in the upper right corner. Several white, wispy clouds are scattered across the sky. In the upper right, there is a large, colorful, abstract shape resembling a mountain or a cloud, composed of various colored segments (purple, red, orange, black, green, blue) with some internal line patterns. The bottom half of the image shows a green landscape. On the left, there is a solid green hill. On the right, there is a larger hill covered in a dense, intricate pattern of small, overlapping geometric shapes (triangles, squares) in various shades of green and yellow.

Using Indexes

ORACLE
Academy

What Are Indexes?

- An index is an optional structure, associated with a table
- Indexes are schema objects that are logically and physically independent of the data in the objects with which they are associated
- You can drop or create an index without physically affecting the indexed table
- The absence or presence of an index does not require a change in the wording of any SQL statement
- An index is a fast access path to a single row of data
 - It affects only the speed of execution

Indexes

- Indexes point to specific information in a table

Table

| Row | EMPNO | ENAME | DEPTNO | JOB | HIREDATE |
|-----|-------|-------|--------|----------|-----------|
| 1 | 7876 | ADAMS | 20 | CLERK | 23-MAY-87 |
| 2 | 7499 | ALLEN | 30 | SALESREP | 20-FEB-81 |
| 3 | 7698 | BLAKE | 30 | MANAGER | 01-MAY-81 |
| 4 | 7782 | CLARK | 10 | MANAGER | 09-JUN-81 |
| 5 | 7902 | FORD | 20 | ANALYST | 03-DEC-81 |

Index on Column
DEPTNO

| DEPTNO | Row |
|--------|-----|
| 10 | 4 |
| 20 | 1,5 |
| 30 | 2,3 |
| 40 | |

ORACLE
Academy

ADS – Section 2
Database Basics – Part 2

Copyright © 2022, Oracle and/or its affiliates. Oracle, Java, and MySQL are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

114

Tables can get very large. Indexing is the mechanism databases use to make sure tables that have a lot of information can provide that information to those that need it very fast. In this example, we have an index on the department number of the table. This will make data retrieval from the table based on department number conditions a lot faster.

Indexes

- Creating an index

```
CREATE INDEX emp_department_ix ON employees (department_id);
```

↑
Index Name

↑
Indexed column on employee table

```
SELECT last_name department_id  
FROM employees  
WHERE department_id = 20;
```

- Was there an index there already?

```
DROP INDEX emp_department_ix;
```

To create an index use the create index statement, with the index name, the table to be indexed and the column in the table to be indexed. In this example we are creating an index called emp_department_ix on the department_id column of the employees table.

Composite (Concatenated) Indexes

- A composite index, also called a concatenated index, is an index on multiple columns in a table

```
CREATE INDEX employees_ix  
ON employees (last_name, department_id, salary);
```

- 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
- The order of the columns used in the definition is important
- In general, the most commonly accessed columns go first

Indexes

- Typically developers index columns for three major reasons:
 - To enforce unique values within a column
 - To improve data access performance, the indexed columns are queried frequently and return a small percentage of the total number of rows in the table
 - To prevent lock escalation when updating rows of tables that use declarative referential integrity
- When a table is created and a **PRIMARY KEY** is specified, an index is automatically created to enforce the primary key constraint
- If you specify **UNIQUE** for a column, a unique index is also created

Unique and Non-unique Indexes

- Indexes can be unique or non unique
- **Unique indexes** guarantee that no two rows of a table have duplicate values in the key column or columns
 - For example, an application may require that no two employees have the same employee ID
- **Non-unique indexes** permit duplicates values in the indexed column or columns
 - For example, the first name column of the employees table may contain multiple Mike values

Indexes – What's involved?

- Creating indexes manually often requires deep knowledge of the data model, application, and data distribution
- As the data changes, you must revisit previous decisions about indexes
 - An index might stop being useful, or new indexes might be required
- Indexes occupy disk space
- The database must update the index when DML occurs on the indexed data, which creates performance overhead

Suggestions on When to Use Indexes

- It is typically good form to index foreign keys, foreign keys are columns in a table that reference another table

```
CREATE INDEX employee_dept_no_fk_idx ON employees (department_id);
```

- The EMPLOYEE table will be frequently searched by the NAME column
- To improve the performance searches and to ensure uniqueness we can create a unique index on the EMPLOYEE table NAME column

```
CREATE UNIQUE INDEX employee_ename_idx ON employees (last_name);
```

When to use indexes?

When a table will be frequently searched by a particular column value.

An abstract landscape illustration. The top half features a pale yellow sky with a bright yellow sun in the upper right corner. Several white, wispy clouds are scattered across the sky. In the upper right, there is a large, colorful, abstract shape resembling a mountain or a cloud, composed of various colored segments (purple, red, orange, black, green, blue) with some internal line patterns. The bottom half of the image shows a green landscape. On the left, there is a solid green hill. On the right, there is a larger hill covered in a dense, repeating pattern of small, interlocking geometric shapes (triangles and squares) in shades of green and yellow.

Using Constraints

ORACLE
Academy

Including Constraints

- Constraints enforce rules at the table level
- Constraints ensure the consistency and integrity of the database
- The following constraint types are valid:
 - NOT NULL
 - UNIQUE
 - PRIMARY KEY
 - FOREIGN KEY
 - CHECK



Data Integrity Constraints

| Constraints | Description |
|--------------------|--|
| NOT NULL | The column cannot contain a null value |
| UNIQUE | The values for a column or a combination of columns must be unique for all rows in the table |
| PRIMARY KEY | The column (or a combination of columns) must contain the unique AND IS NOT NULL value for all rows |
| FOREIGN KEY | The column (or a combination of columns) must establish and enforce a reference to a column or a combination of columns in another (or the same) table |
| CHECK | A condition must be true |

Constraint Guidelines

- Column-level constraints are included when the column is defined
- Table-level constraints are defined at the end of the table definition, and must refer to the column or columns on which the constraint pertains
- Functionally, a column-level constraint is the same as a table-level constraint
- NOT NULL constraints can be defined only at the column level
- Constraints that apply to more than one column must be defined at the table level

Defining Constraints

- CREATE TABLE with CONSTRAINTS syntax:

```
CREATE TABLE [schema.]table
    (column datatype [DEFAULT expr]
    [column_constraint],
    ...
    [table_constraint][,...]);
```

- Constraints can be defined when creating a table by using the column_constraint or table_constraint clauses

Creating Tables With Constraints

- You can specify rules for each column of a table called integrity constraints
- One example is a NOT NULL integrity constraint
 - This constraint forces the column to contain a value in every row

```
CREATE TABLE dept(  
    deptno      NUMBER(2) ,  
    dname       VARCHAR2(14) NOT NULL ,  
    loc         VARCHAR2(13) ,  
    create_date DATE DEFAULT SYSDATE ,  
    CONSTRAINT pk_departments PRIMARY KEY(deptno)  
);
```

In the create table in this example, the dname column cannot be null.

Defining Constraints

- Column-level constraint syntax at column definition:

```
column [CONSTRAINT constraint_name] constraint_type,
```

- Table-level constraint syntax at end of table definition:

```
column,...  
  [CONSTRAINT constraint_name] constraint_type  
  (column, ...),
```

Table constraints are usually defined at the end of a table definition statement.

Examples: Defining Constraints

- Defining PRIMARY KEY constraints
 - Column level constraint that is a primary key as part of the column definitions

```
CREATE TABLE employees1(  
    employee_id NUMBER(6) CONSTRAINT emp_id_pk PRIMARY KEY,  
    first_name VARCHAR2(20)  
);
```

- Table level constraint that is also a primary key constraint on the same column

```
CREATE TABLE employees2(  
    employee_id NUMBER(6),  
    first_name VARCHAR2(20),  
    CONSTRAINT emp_id_pk PRIMARY KEY (employee_id)  
);
```

These two examples basically achieve the same thing.

NOT NULL Constraint

- NOT NULL constraints can be defined ONLY at the column level:


```
CREATE TABLE employees3(  
  employee_id    NUMBER(6) ,  
  last_name      VARCHAR2(25) NOT NULL,  
  email          VARCHAR2(25) ,  
  salary         NUMBER(8,2) ,  
  commission_pct NUMBER(2,2) ,  
  hire_date      DATE CONSTRAINT hire_date_nn NOT NULL  
);
```

UNIQUE Constraint


- A UNIQUE key integrity constraint requires that every value in a column or a set of columns be unique
- If the UNIQUE constraint has more than one column, that group of columns is called a composite unique key
- UNIQUE constraints enable the input of nulls
- A null in a column (or in all columns of a composite UNIQUE key) always satisfies a UNIQUE constraint

UNIQUE Constraint


EMPLOYEES

UNIQUE constraint 


| EMPLOYEE_ID | LAST_NAME | EMAIL |
|-------------|-----------|----------|
| 100 | King | SKING |
| 101 | Kochhar | NKOCHHAR |
| 102 | De Haan | LDEHAAN |
| 200 | Whalen | JWHALEN |
| 205 | Higgins | SHIGGINS |
| ... | | |

 **INSERT INTO**

| | | |
|-----|-------|--------|
| 208 | Smith | JSMITH |
|-----|-------|--------|

 **Allowed**

| | | |
|-----|-------|--------|
| 209 | Smith | JSMITH |
|-----|-------|--------|

 **Not allowed: already exists**

In this example we have a unique constraint on the email column. We perform two inserts of records with emails JSMITH. The first insert will perform correctly. However, the second insert will fail because the previous insert already used the JSMITH email value and so the unique constraint is violated.

Finding Out What Constraints Exist on Tables?

- Query the user_cons_columns with or without the WHERE clause

```
SELECT *  
FROM user_cons_columns  
WHERE table_name = '<your table name>';
```

- Display the constraints on the table we just created

```
SELECT *  
FROM user_cons_columns;
```

To determine which constraints exist on tables Query the user_cons_columns with or without the WHERE clause. This is important to know because sometimes you will want to manipulate data in a table and a constraint will prevent you from doing so. Finding out what constraints exist on a table will help get around this problem.

PRIMARY KEY Constraint

- A **PRIMARY KEY** constraint creates a primary key for the table
- **Only one primary key** can be created for each table
- **Uniqueness** is part of the primary key constraint definition (no duplicate rows)
- Implicitly creates a unique index on the primary key column or columns
- The PRIMARY KEY can be a set of columns (composite primary key)
- No column that is part of the primary key can contain a null value

PRIMARY KEY Constraint

DEPARTMENTS  PRIMARY KEY

| DEPARTMENT_ID | DEPARTMENT_NAME | MANAGER_ID | LOCATION_ID |
|---------------|-----------------|------------|-------------|
| 10 | Administration | 200 | 1700 |
| 20 | Marketing | 201 | 1800 |
| 50 | Shipping | 124 | 1500 |
| 60 | IT | 103 | 1400 |

Not allowed
(null value) 

 INSERT INTO

| | | | |
|------|-------------------|-----|------|
| NULL | Public Accounting | 124 | 2500 |
| 50 | Finance | 124 | 1500 |

 Not allowed (50 already exists)

ORACLE
Academy

ADS – Section 2
Database Basics – Part 2

Copyright © 2022, Oracle and/or its affiliates. Oracle, Java, and MySQL are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

134

In this example the department_id column is the primary key for the table. When we try to insert Public Accounting with a department_id of NULL into the table the operation fails because a primary key cannot be NULL.
When we try to insert Finance with a department_id of 50, the operation also fails because a department_id of 50 already exists in the table.

FOREIGN KEY Constraint

- The FOREIGN KEY (or referential integrity) constraint designates a column or a combination of columns as a foreign key
- Establishes a relationship with a primary key in the same table or a different table
- Here are the guidelines for foreign key constraints:
 - A foreign key value must match an existing value in the parent table or be NULL
 - Foreign keys are based on data values and are purely logical, rather than physical, pointers



FOREIGN KEY Constraint: Keywords

- **FOREIGN KEY:** Defines the column in the child table at the table-constraint level
- **REFERENCES:** Identifies the table and column in the parent table
- **ON DELETE CASCADE:** Deletes the dependent rows in the child table when a row in the parent table is deleted
- **ON DELETE SET NULL:** Converts dependent foreign key values to null



FOREIGN KEY Constraint

DEPARTMENTS

| DEPARTMENT_ID | DEPARTMENT_NAME | MANAGER_ID | LOCATION_ID |
|---------------|-----------------|------------|-------------|
| 10 | Administration | 200 | 1700 |
| 20 | Marketing | 201 | 1800 |
| 50 | Shipping | 124 | 1500 |

EMPLOYEES

| EMPLOYEE_ID | SALARY | DEPARTMENT_ID |
|-------------|--------|---------------|
| 100 | 24000 | 90 |
| 101 | 17000 | 90 |
| 102 | 17000 | 90 |

FOREIGN KEY



INSERT INTO

| | | |
|-----|------|---|
| 200 | Ford | 9 |
|-----|------|---|

Not allowed
(9 does not exist)

| | | |
|-----|------|----|
| 200 | Ford | 50 |
|-----|------|----|

Allowed

ORACLE
Academy

ADS – Section 2
Database Basics – Part 2

Copyright © 2022, Oracle and/or its affiliates. Oracle, Java, and MySQL are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

137

In this case we are trying to add employee id 200 to the employee table with a department id of 9. The operation fails because the department_id in the foreign key table does not have department 9 in it. When we try to insert the same employee but with department 50, the operation completes successfully.

FOREIGN KEY Constraint Defined at the Column Level

- FOREIGN KEY constraints can be defined at the column level

```
CREATE TABLE employees (  
    employee_id      NUMBER(6) ,  
    last_name        VARCHAR2(25) ,  
    email            VARCHAR2(25) ,  
    salary            NUMBER(8,2) ,  
    commission_pct   NUMBER(2,2) ,  
    hire_date        DATE ,  
    department_id    NUMBER(4) CONSTRAINT emp_dept_fk  
                    REFERENCES departments(department_id)  
);
```

FOREIGN KEY Constraint Defined at the Table Level

- FOREIGN KEY constraints can also be defined at the table level

```
CREATE TABLE employees (  
    employee_id      NUMBER(6) ,  
    last_name        VARCHAR2(25) ,  
    email            VARCHAR2(25) ,  
    salary            NUMBER(8,2) ,  
    commission_pct   NUMBER(2,2) ,  
    hire_date         DATE ,  
    department_id     NUMBER(4) ,  
    CONSTRAINT emp_dept_fk FOREIGN KEY (department_id)  
        REFERENCES departments(department_id)  
);
```

PRIMARY and FOREIGN KEY Constraints

- Here is an example of two tables related by a primary key and a foreign key constraint

```
CREATE TABLE publishers(  
  id          NUMBER(2),  
  name        VARCHAR2(100) NOT NULL,  
  CONSTRAINT plr_id_pk PRIMARY KEY (ID)  
);
```

```
CREATE TABLE books(  
  id          VARCHAR2(6),  
  title       VARCHAR2(255) NOT NULL,  
  publisher_id NUMBER(2),  
  author_id   NUMBER(3),  
  CONSTRAINT bok_id_pk PRIMARY KEY (ID),  
  CONSTRAINT bok_plr_fk FOREIGN KEY (publisher_id)  
    REFERENCES publishers(id)  
);
```

The relationship is the published id, which must be unique in the publishers table and must exist in that table before a book with that published id can be inserted into the books table

CHECK Constraint

- It defines a condition that each row must satisfy
- It cannot reference columns from other tables

```
CREATE TABLE employees (  
    ...  
    salary          NUMBER(8,2) CONSTRAINT emp_salary_min  
                    CHECK (salary > 0),  
    ...  
);
```

- In this case the salary must be greater than 0

A check constraint defines a condition that each row must satisfy in order for data to be inserted into that column.

CHECK Constraint Example

- The hire date must be after Jan 1, 2018

```
CREATE TABLE employees(  
    employee_id      NUMBER(6),  
    last_name        VARCHAR2(25),  
    email            VARCHAR2(25),  
    salary            NUMBER(8,2),  
    commission_pct   NUMBER(2,2),  
    hire_date        DATE,  
    ...  
    CONSTRAINT hire_date_min CHECK  
        (hire_date > '01-JAN-2022')  
);
```

In this case we are checking that the hire_date for the employee is after January 1, 2022.

Add Constraints with ALTER Table

- You can use the ALTER TABLE statement to add constraints after a table was created
- In this case, we can use the ALTER statement to add a constraint to the hire_date column of an existing employees table to check to make sure the hire date is after January 1, 2022

```
ALTER TABLE employees(  
  ADD CONSTRAINT  
  hire_date_min CHECK hire_date > '01-JAN-2022')  
);
```

We previously discussed that we could modify tables after creation with the ALTER statement.

An abstract landscape illustration. The top half features a pale yellow sky with a bright yellow sun in the upper right corner. Below the sun is a colorful, multi-colored mountain range with shades of purple, red, orange, and black. To the left of this range is a white, stylized mountain range. The bottom half of the image shows a green hill on the left and a green hill on the right, both with a pattern of small, white, interlocking geometric shapes. The title 'Joining Tables' is centered in the upper half of the image.

Joining Tables

ORACLE
Academy

Table JOINS

- Many database operations require information that resides in more than one table
- These operations only require some information from each table
- The appropriate information from each table must be selected however the corresponding information must correlate in some way
- JOIN operations provide this functionality



Obtaining Data From Multiple Tables


- Often data is needed from more than one table
- To produce a report, you need to link the EMPLOYEES and DEPARTMENTS tables, and access data from both tables:

EMPLOYEES

| EMPLOYEE_ID | LAST_NAME | DEPARTMENT_ID |
|-------------|-----------|---------------|
| 100 | King | 90 |
| 149 | Zlotkey | 80 |
| 103 | Ernst | 60 |

DEPARTMENTS

| DEPARTMENT_ID | DEPARTMENT_NAME | LOCATION_ID |
|---------------|-----------------|-------------|
| 60 | IT | 1400 |
| 80 | Sales | 2500 |
| 90 | Executive | 1700 |



| EMPLOYEE_ID | DEPARTMENT_ID | DEPARTMENT_NAME |
|-------------|---------------|-----------------|
| 100 | 90 | Executive |
| 149 | 80 | Sales |
| 102 | 60 | IT |

In this example, to produce a report of all employees, their employee id, department id, and department name, the departments and employees table must be joined

Types of Joins

- Natural join with the NATURAL JOIN clause
- Join with the USING Clause
- Join with the ON Clause
- Equijoin of Inner Join
- OUTER joins:
 - LEFT OUTER JOIN
 - RIGHT OUTER JOIN
 - FULL OUTER JOIN
- CROSS JOIN



SQL defines several ways to join tables including natural joins, outer joins, inner joins, cross joins and other. Each yields a different result of the table combination.

INNER Versus OUTER Joins

- In SQL:1999, the join of two tables returning only matched rows is called an INNER join (NATURAL JOIN, USING, ON clauses)
- A join between two tables that returns the results of the INNER join as well as the unmatched rows from the left (or right) table is called a left (or right) OUTER join
- A join between two tables that returns the results of an INNER join as well as the results of left and right OUTER join is a full OUTER join



Use a Join to Query Data From More Than One Table:

```
SELECT table1.column, table2.column
FROM   table1
[NATURAL JOIN table2] |
[JOIN table2 USING (column_name)] |
[JOIN table2
  ON (table1.column_name = table2.column_name)] |
[LEFT|RIGHT|FULL OUTER JOIN table2
  ON (table1.column_name = table2.column_name)] |
[CROSS JOIN table2];
```

- **table1.column** denotes the table and the column from which data is retrieved
- **NATURAL JOIN** joins two tables based on the same column name
- **JOIN table2 USING column_name** performs an equijoin based on the column name
- **JOIN table2 ON table1.column_name = table2.column_name** performs an equijoin based on the condition in the **ON** clause
- **LEFT/RIGHT/FULL OUTER** is used to perform OUTER joins
- **CROSS JOIN** returns a Cartesian product from the two tables

JOIN Example

- Join employees and departments tables where department_id matches

```
SELECT employees.first_name, departments.department_name,  
       departments.manager_id  
FROM   employees JOIN departments  
USING  (department_id);
```

| FIRST_NAME | DEPARTMENT_NAME | MANAGER_ID |
|------------|-----------------|------------|
| Jennifer | Administration | 200 |
| Michael | Marketing | 201 |

Qualifying Ambiguous Column Names

- Use table prefixes to qualify column names that are in multiple tables, avoiding ambiguity
- Instead of full table name prefixes, use table aliases
- Table alias gives a table a shorter name, keeps SQL code smaller, uses less memory

Qualifying Ambiguous Column Names

- Use table aliases to distinguish columns that have identical names, but in different tables
- The table name is specified in full, followed by a space, and the table alias

```
SELECT e.first_name, d.department_name, d.manager_id
FROM   employees e JOIN departments d
USING  (department_id);
```

| FIRST_NAME | DEPARTMENT_NAME | MANAGER_ID |
|------------|-----------------|------------|
| Jennifer | Administration | 200 |
| Michael | Marketing | 201 |

Creating Natural Joins

- The NATURAL JOIN clause is based on all columns in the two tables that have the same name and the same data type
- It selects rows from the two tables that have equal values in all matched columns
- If columns with the same names have different data types, an error is returned

Retrieving Records With Natural Joins

- Uses the only field which is common to both tables - DEPARTMENT_ID to do the join

```
SELECT department_id, department_name, location_id, city
FROM departments NATURAL JOIN locations;
```

| DEPARTMENT_ID | DEPARTMENT_NAME | LOCATION_ID | CITY |
|---------------|-----------------|-------------|---------------------|
| 20 | Marketing | 1800 | Toronto |
| 80 | Sales | 2500 | Oxford |
| 60 | IT | 1400 | Southlake |
| 50 | Shipping | 1500 | South San Francisco |
| 10 | Administration | 1700 | Seattle |
| 90 | Executive | 1700 | Seattle |

Here is an example of a natural join. It uses the only field which is common to both tables, DEPARTMENT_ID, to do the join.

Creating Joins With the USING Clause

- By default if multiple columns are shared by the tables being joined all common fields are used in the join
- Use the USING clause to specify a single column for the JOIN instead of a NATURAL JOIN
- The USING clause can also be used to match columns that have the same name but different data types
- The NATURAL JOIN and USING clauses are mutually exclusive

Creating Joins With the USING Clause

- Values in the join column (DEPARTMENT_ID) column in both the tables must be equal

EMPLOYEES

| EMPLOYEE_ID | LAST_NAME | DEPARTMENT_ID |
|-------------|-----------|---------------|
| 100 | King | 90 |
| 200 | Whalen | 10 |
| 205 | Higgins | 110 |
| 206 | Gietz | 110 |
| 149 | Zlotkey | 80 |
| 124 | Mourgos | 50 |

DEPARTMENTS

| DEPARTMENT_ID | DEPARTMENT_NAME | LOCATION_ID |
|---------------|-----------------|-------------|
| 10 | Administration | 1700 |
| 50 | Shipping | 1500 |
| 60 | IT | 1400 |
| 80 | Sales | 2500 |
| 90 | Executive | 1700 |
| 110 | Accounting | 1700 |

Foreign key

Primary key

Retrieving Records With the USING Clause

- The USING clause specifies that the join is done with the DEPARTMENT_ID

```
SELECT employee_id, last_name, location_id, department_id
FROM   employees JOIN departments
      USING (department_id);
```

| EMPLOYEE_ID | LAST_NAME | LOCATION_ID | DEPARTMENT_ID |
|-------------|-----------|-------------|---------------|
| 200 | Whalen | 1700 | 10 |
| 201 | Hartstein | 1800 | 20 |
| 202 | Fay | 1800 | 20 |
| 124 | Mourgos | 1500 | 50 |

Here is an example where we are joining two tables with the USING clause specifying the department_id column as the common joining column between the tables.

Creating Joins With the ON Clause

- Use the ON clause to specify arbitrary conditions or columns to join
- The join condition is separated from other search conditions
- A USING clause creates an equijoin between two tables using one column with the same name, regardless of the data type
- An ON clause creates an equijoin between two tables using one column from each table, regardless of the name or data type

Retrieving Records With the ON Clause

- You can use the ON clause to join columns that have different names or data types

```
SELECT employee_id, last_name, department_name, location_id
FROM employees e JOIN departments d
ON d.department_id = e.department_id;
```

| EMPLOYEE_ID | LAST_NAME | DEPARTMENT_ID | DEPARTMENT_ID | LOCATION_ID |
|-------------|-----------|---------------|---------------|-------------|
| 200 | Whalen | 10 | 10 | 1700 |
| 201 | Hartstein | 20 | 20 | 1800 |
| 202 | Fay | 20 | 20 | 1800 |
| 124 | Mourgos | 50 | 50 | 1500 |
| 141 | Rajs | 50 | 50 | 1500 |
| 142 | Davies | 50 | 50 | 1500 |

In this example, we specify the department_id columns of both tables as the columns to join on by using the ON clause.

Retrieving Data From 3 Tables With the ON Clause

- There must be 2 join statements when joining 3 tables as shown:

```
SELECT employee_id, city, department_name
FROM   employees e JOIN departments d
ON     d.department_id = e.department_id
JOIN   locations l
ON     d.location_id = l.location_id;
```

| EMPLOYEE_ID | CITY | DEPARTMENT_NAME |
|-------------|-----------|-----------------|
| 201 | Toronto | Marketing |
| 202 | Toronto | Marketing |
| 149 | Oxford | Sales |
| 174 | Oxford | Sales |
| 176 | Oxford | Sales |
| 103 | Southlake | IT |

```
DESCRIBE employees;
```

```
DESCRIBE departments;
```

```
DESCRIBE locations;
```

Multiple ON clauses may be used on the same select statement if more than 2 tables are to be joined. In this example we are joining three tables, employees, departments and locations and we are applying join conditions on pairs of tables on their matching columns.

Returning Records With No Direct Match

- Using OUTER Joins

DEPARTMENTS

| DEPARTMENT_NAME | DEPARTMENT_ID |
|-----------------|---------------|
| Administration | 10 |
| Marketing | 20 |
| Shipping | 50 |
| IT | 60 |
| Sales | 80 |
| Executive | 90 |
| Accounting | 110 |
| Contracting | 190 |

There are no employees
in department 190

Employee "Grant" has not been
assigned a department ID.

Equijoin with EMPLOYEES

| DEPARTMENT_ID | LAST_NAME |
|---------------|-----------|
| 90 | King |
| 90 | Kochhar |
| 90 | De Haan |
| 10 | Whalen |
| 80 | Taylor |
| - | Grant |
| 50 | Mourgos |
| 20 | Fay |

ORACLE
Academy

ADS – Section 2
Database Basics – Part 2

Copyright © 2022, Oracle and/or its affiliates. Oracle, Java, and MySQL are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

161

If a row does not satisfy a join condition, the row does not appear in the query result. In the slide example, a simple equijoin condition is used on the EMPLOYEES and DEPARTMENTS tables to return the result on the right.

```
SELECT employees.department_id, department_name, last_name
FROM employees, departments
WHERE employees.department_id = departments.department_id;
```

The result set does not contain:

- Department ID 190, because there are no employees with that department ID recorded in the EMPLOYEES table
- The employee with the last name of Grant, because this employee has not been assigned a department ID

To return the department record that does not have any employees, or employees that do not have an assigned department, you can use an OUTER join.

LEFT OUTER JOIN

- Select all employee (left table) records even if they are not assigned to a department

```
SELECT e.last_name, e.department_id, d.department_name
FROM employees e LEFT OUTER JOIN departments d
ON (e.department_id = d.department_id);
```

| LAST_NAME | DEPARTMENT_ID | DEPARTMENT_NAME |
|-----------|---------------|-----------------|
| Whalen | 10 | Administration |
| Fay | 20 | Marketing |
| Hartstein | 20 | Marketing |
| Vargas | 50 | Shipping |
| Matos | 50 | Shipping |
| Higgins | 110 | Accounting |
| Grant | - | - |

In this example we are doing a left outer join of employees and department. Therefore Grant, who does not have a department_id still shows up in the result set

RIGHT OUTER JOIN

- Select all department (right table) records even if they have no employees in them

```
SELECT e.last_name, e.department_id, d.department_name
FROM employees e LEFT OUTER JOIN departments d
ON (e.department_id = d.department_id);
```

| LAST_NAME | DEPARTMENT_ID | DEPARTMENT_NAME |
|-----------|---------------|-----------------|
| Whalen | 10 | Administration |
| Hartstein | 20 | Marketing |
| Fay | 20 | Marketing |
| Mourgos | 50 | Shipping |
| Rajs | 50 | Shipping |
| Davies | 50 | Shipping |
| - | - | Contracting |

The right outer join does the exact opposite of the left outer join. It selects all department (right table) records even if they have no employees in them.

FULL OUTER JOIN

- Select all employee records and all department records
- Returns the combination of left and right outer joins

```
SELECT e.last_name, e.department_id, d.department_name
FROM employees e FULL OUTER JOIN departments d
ON (e.department_id = d.department_id);
```

| LAST_NAME | DEPARTMENT_ID | DEPARTMENT_NAME |
|-----------|---------------|-----------------|
| King | 90 | Executive |
| Kochhar | 90 | Executive |
| Taylor | 80 | Sales |
| Grant | - | - |
| Mourgos | 50 | Shipping |
| Fay | 20 | Marketing |
| - | - | Contracting |

Cartesian Products

- A Cartesian product is when all combinations of rows are displayed
 - All rows in the first table are joined to all rows in the second table (rarely useful)

```
SELECT last_name, department_name  
FROM employees, departments;
```

- A Cartesian product is formed when a join condition is omitted or invalid
- Always include a valid join condition if you want to avoid a Cartesian product

```
SELECT last_name, department_name  
FROM employees e, departments d  
WHERE e.department_id = d.department_id;
```

A Cartesian product tends to generate a large number of rows, and the result is rarely useful except for some tests when you need to generate a large number of rows to simulate a reasonable amount of data.

Generating a Cartesian Product

EMPLOYEES (40 rows)

| EMPLOYEE_ID | LAST_NAME | DEPARTMENT_ID |
|-------------|-----------|---------------|
| 100 | King | 90 |
| 149 | Zlotkey | 80 |
| 103 | Ernst | 60 |
| ... | | |

DEPARTMENTS (9 rows)

| DEPARTMENT_ID | DEPARTMENT_NAME | LOCATION_ID |
|---------------|-----------------|-------------|
| 60 | IT | 1400 |
| 80 | Sales | 2500 |
| 90 | Executive | 1700 |
| ... | | |

Cartesian product:
40 x 9 = 360 rows

| EMPLOYEE_ID | LAST_NAME | DEPARTMENT_ID | DEPARTMENT_NAME | LOCATION_ID |
|-------------|-----------|---------------|-----------------|-------------|
| 100 | King | 90 | Administration | 1700 |
| 101 | Kochhar | 90 | Administration | 1700 |
| 102 | De Haan | 90 | Administration | 1700 |
| 200 | Whalen | 10 | Administration | 1700 |
| 205 | Higgins | 110 | Administration | 1700 |
| 206 | Gietz | 110 | Administration | 1700 |
| 149 | Zlotkey | 80 | Administration | 1700 |
| ... | | | | |

ORACLE
Academy

ADS – Section 2
Database Basics – Part 2

Copyright © 2022, Oracle and/or its affiliates. Oracle, Java, and MySQL are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

166

Here is an example of a Cartesian join of a table with 40 rows and a table with 9 rows. The result set is 360 rows.

An abstract landscape illustration. The top half features a pale yellow sky with a bright yellow sun in the upper right corner. Below the sky, there are stylized mountains: one on the left with white concentric line patterns, and a larger one on the right with a colorful patchwork of purple, red, black, and orange. The bottom half of the image shows green hills. The left hill is a solid teal color, while the right hill is filled with a dense, intricate pattern of small, overlapping geometric shapes in various shades of green.

Lab 2: Tables, Keys, SQL Basics

ORACLE
Academy

Summary

- In this lesson, you should have learned:
 - Database Refresh/SQL
 - Creating Databases, Inserting, and Updating Data
 - Retrieving/Conditionally Retrieving Data
 - Ordering Data, Using Indexes
 - Joining Tables
 - Lab 2



