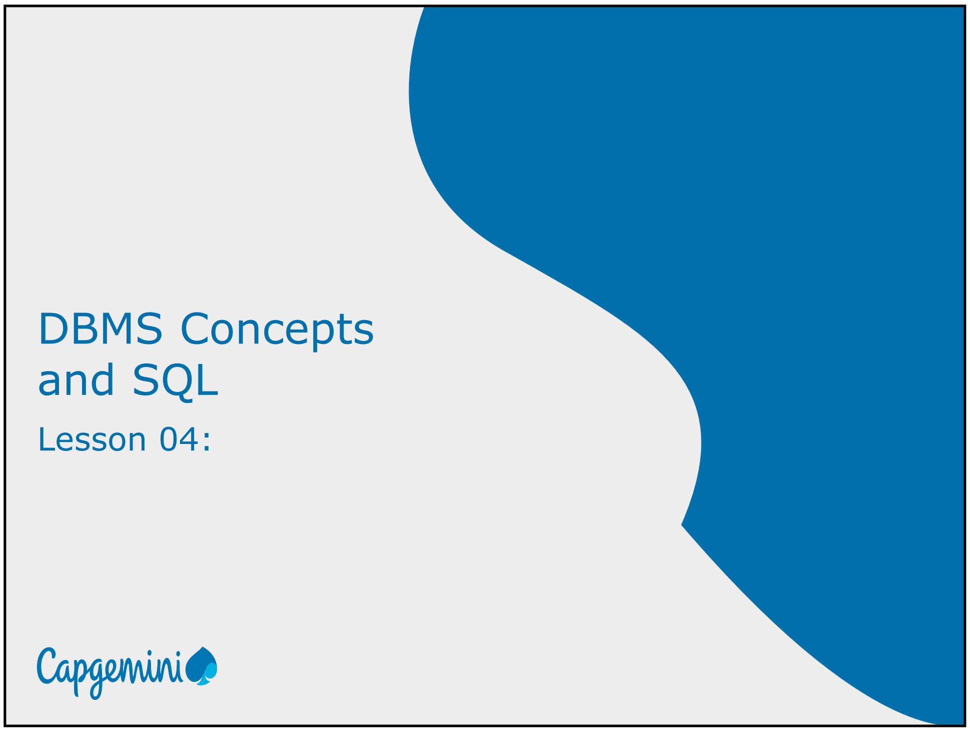
Instructor Notes:

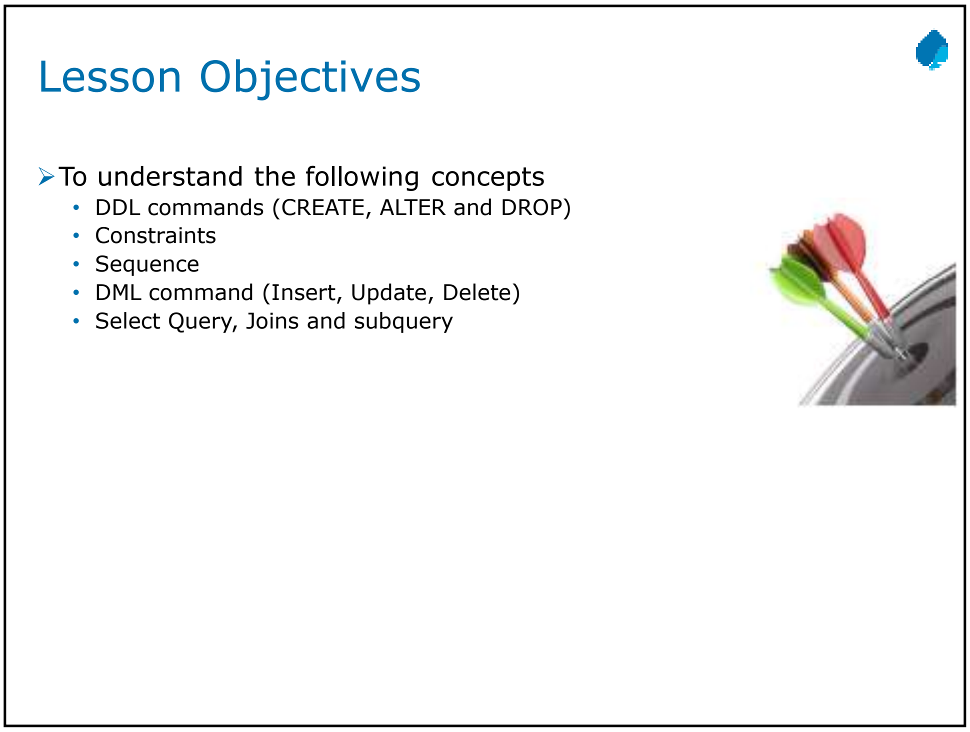


DBMS Concepts and SQL

Lesson 04:

Capgemini

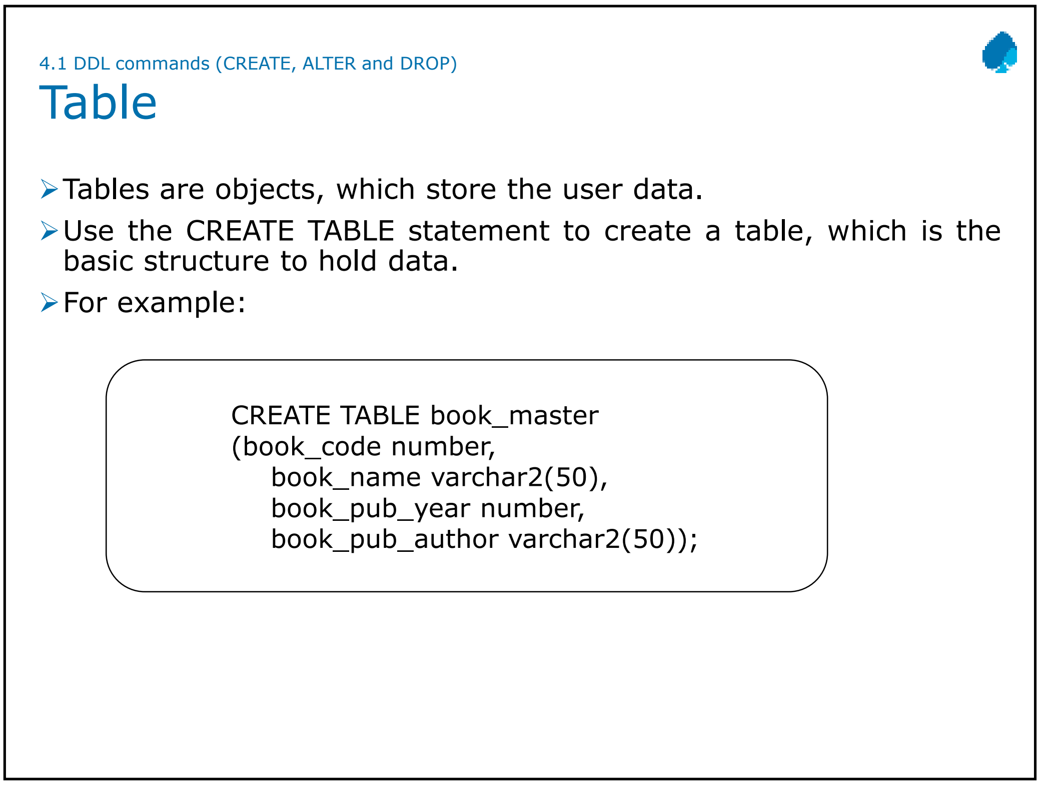
Instructor Notes:



Lesson Objectives

➢ To understand the following concepts
- DDL commands (CREATE, ALTER and DROP)
- Constraints
- Sequence
- DML command (Insert, Update, Delete)
- Select Query, Joins and subquery

Instructor Notes:



4.1 DDL commands (CREATE, ALTER and DROP)

Table

- Tables are objects, which store the user data.
- Use the CREATE TABLE statement to create a table, which is the basic structure to hold data.
- For example:

```
CREATE TABLE book_master
(book_code number,
    book_name varchar2(50),
    book_pub_year number,
    book_pub_author varchar2(50));
```

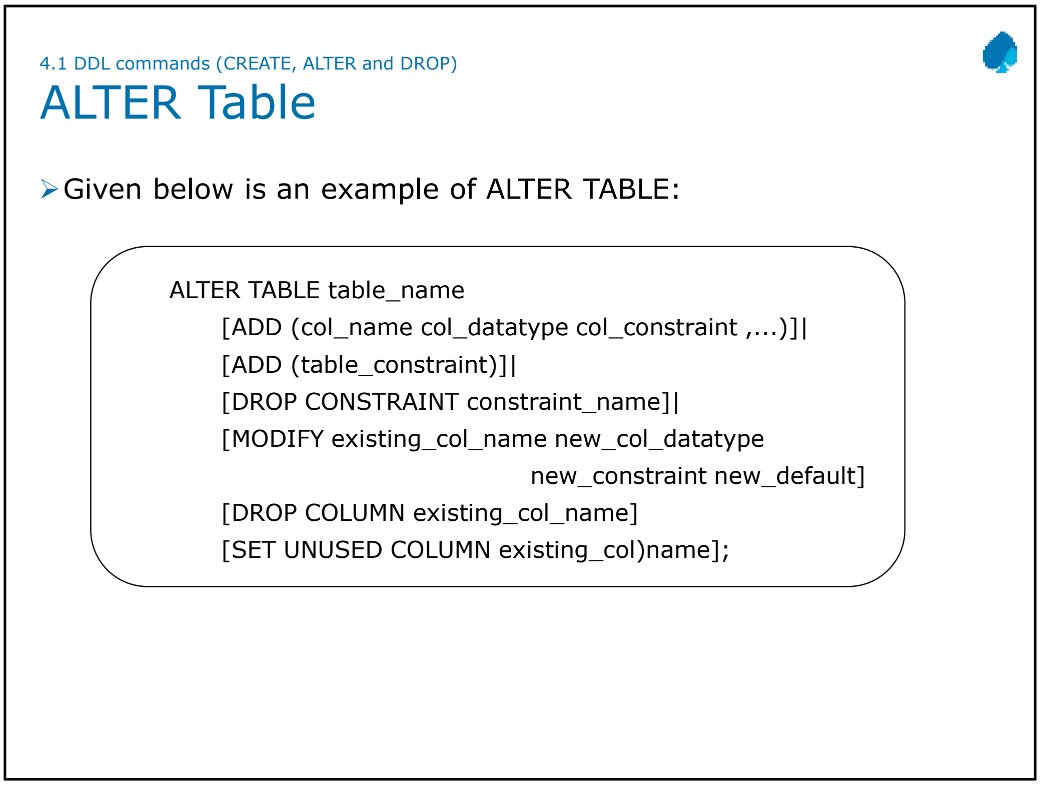
Creating tables is done with the create table command. You can add rows to a table with the INSERT statement, after creating a table.
The above create table command does the following:
- Defines the table name
- Defines the columns in the table and the datatypes of those columns

In above example, we create a table called BOOK_MASTER which has 4 columns. The first column and third column is defined as NUMBER datatype. This means we will be storing numbers in this column. The second and fourth columns are of VARCHAR2 datatype. We will be storing text data in these columns
Syntax:

```
CREATE TABLE table_name
(
{col_name.col_datatype [[CONSTRAINT
const_name][col_constraint]]},...
[table_constraint],...
)
[AS query]
```

- If a table is created as shown in the slide, then there is no restriction on the data that can be stored in the table.
- However, if we wish to put some restriction on the data, which can be stored in the table, then we must supply some “constraints” for the columns. We will see the Constraints as next topic.

Instructor Notes:



4.1 DDL commands (CREATE, ALTER and DROP)

ALTER Table

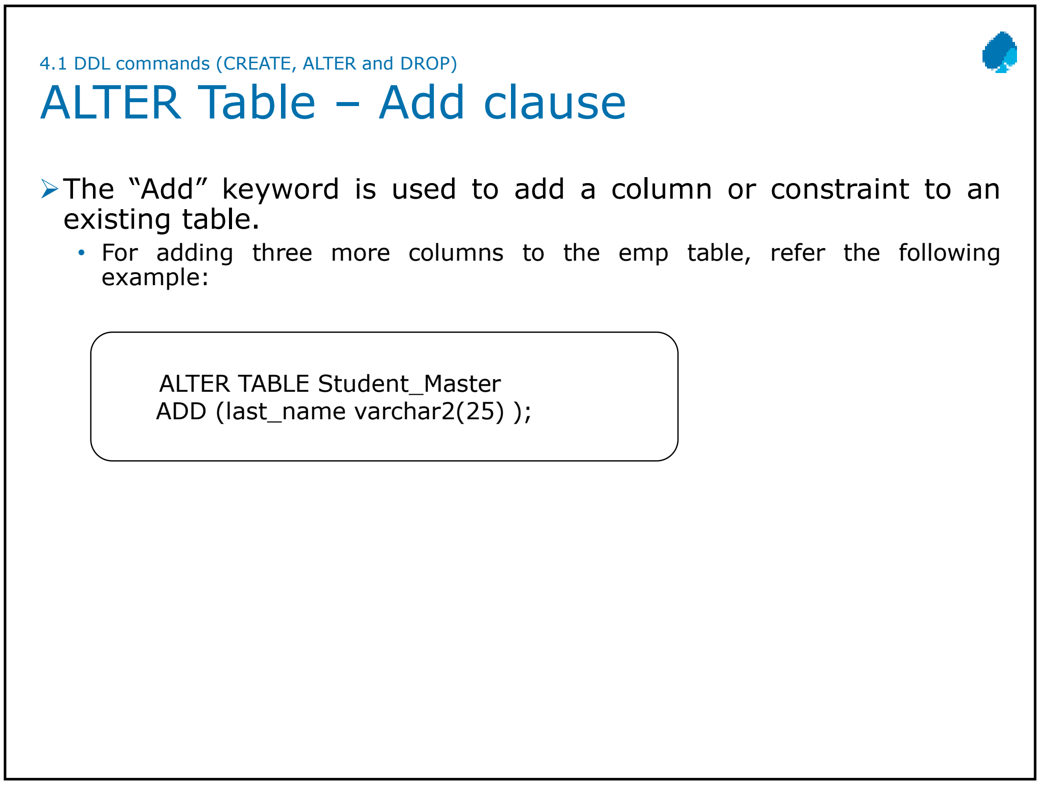
- Given below is an example of ALTER TABLE:

```
ALTER TABLE table_name
        [ADD (col_name col_datatype col_constraint ,...)]|
        [ADD (table_constraint)]|
        [DROP CONSTRAINT constraint_name]|
        [MODIFY existing_col_name new_col_datatype
                                        new_constraint new_default]
        [DROP COLUMN existing_col_name]
        [SET UNUSED COLUMN existing_col)name];
```

Examples of ALTER TABLE:

- Table_name must be an existing table.
- A column can be removed from an existing table by using ALTER TABLE.
- The uses of modifying columns are:
  - Can increase the width of a character column, any time.
  - Can increase the number of digits in a number, any time.
  - Can increase or decrease the number of decimal places in a number column, any time. Any reduction on precision and scale can be on empty columns only.
  - Can add only NOT NULL constraint by using Column constraints. All other constraints have to be specified as Table constraints.

Instructor Notes:



ALTER Table – Add clause

- The “Add” keyword is used to add a column or constraint to an existing table.
  - For adding three more columns to the emp table, refer the following example:

```
ALTER TABLE Student_Master
ADD (last_name varchar2(25) );
```

ALTER TABLE – Add clause:

- The ADD clause allows to add a column or constraint. You can also add multiple columns in one statement separated by comma.
- A column with constraints can also be added as shown in the following example:

```
ALTER TABLE Department_Master
ADD (dept_name varchar2(10) NOT NULL);
```

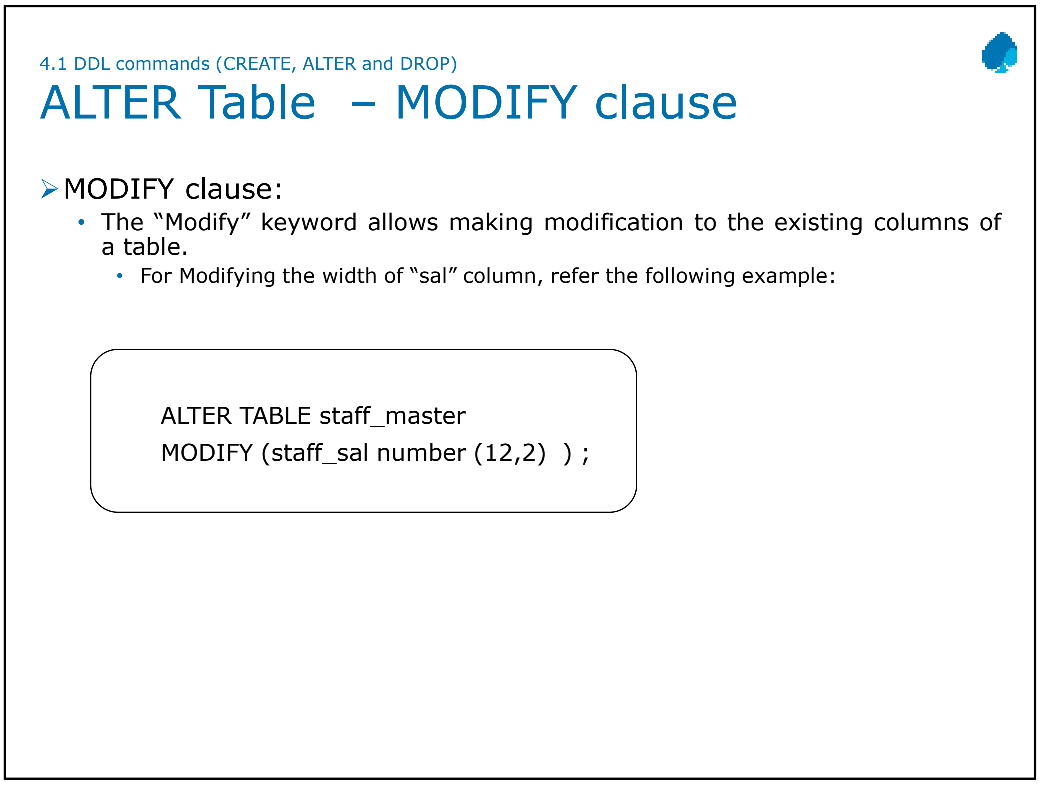
Instructor Notes:

ALTER Table – Add clause

- For adding Referential Integrity on “mgr_code" column, refer the following example:

```
 ALTER TABLE staff_master
    ADD CONSTRAINT FK FOREIGN KEY  (mgr_code)
REFERENCES staff_master(staff_code);
```

Instructor Notes:



4.1 DDL commands (CREATE, ALTER and DROP)

ALTER Table – MODIFY clause

- MODIFY clause:
  - The "Modify" keyword allows making modification to the existing columns of a table.
    - For Modifying the width of "sal" column, refer the following example:

```
ALTER TABLE staff_master
MODIFY (staff_sal number (12,2)  ) ;
```

<u>ALTER TABLE- Modify Clause</u>

The use of modifying the columns with the Enable | Disable clause are:

- Can increase "column width" of a character any time.
- Can increase the "number of digits" in a number at any time.
- Can increase or decrease the "number of decimal places" in a number column at any time. Any reduction on "precision" and "scale" can only be on empty columns.
- Can only add the NOT NULL constraint by using "column constraints". Rest all other constraints have to be specified as "table constraints".

Instructor Notes:

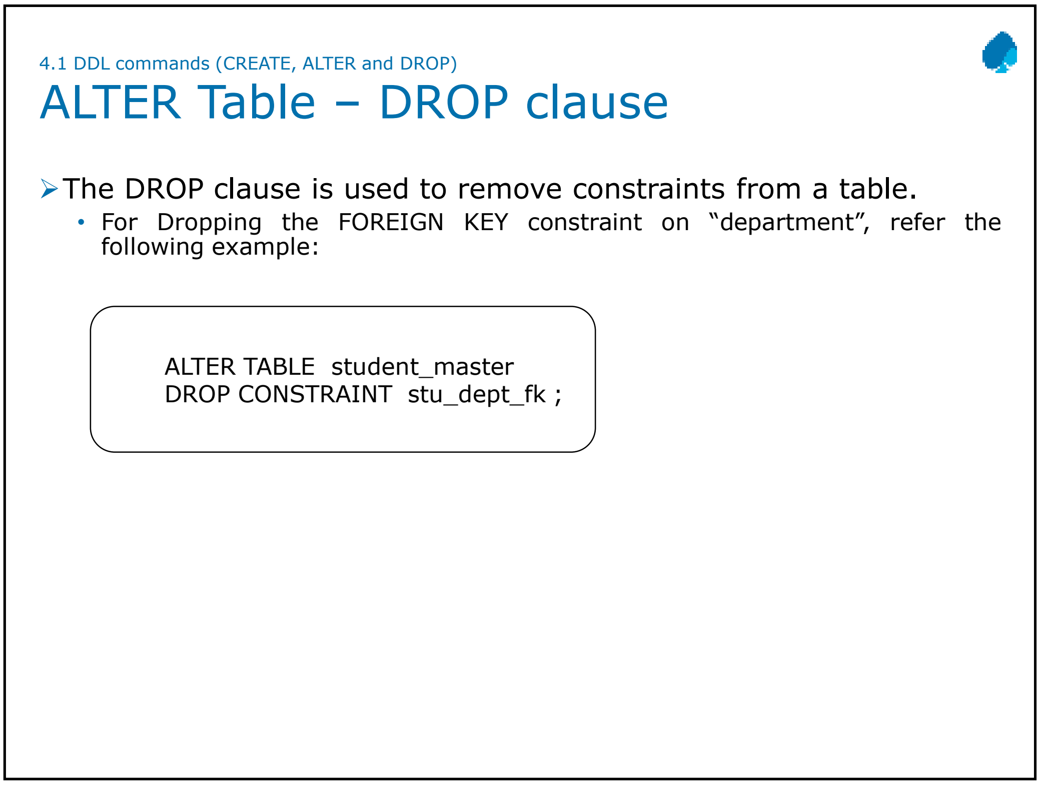
4.1 DDL commands (CREATE, ALTER and DROP)

ALTER Table – Enable | Disable clause

- ENABLE | DISABLE Clause:
  - The ENABLE | DISABLE clause allows constraints to be enabled or disabled according to the user choice without removing them from a table.
  - Refer the following example:

```
ALTER TABLE staff_master DISABLE CONSTRAINT
SYS_C000934;
```

Instructor Notes:



4.1 DDL commands (CREATE, ALTER and DROP)

ALTER Table – DROP clause

- The DROP clause is used to remove constraints from a table.
  - For Dropping the FOREIGN KEY constraint on "department", refer the following example:

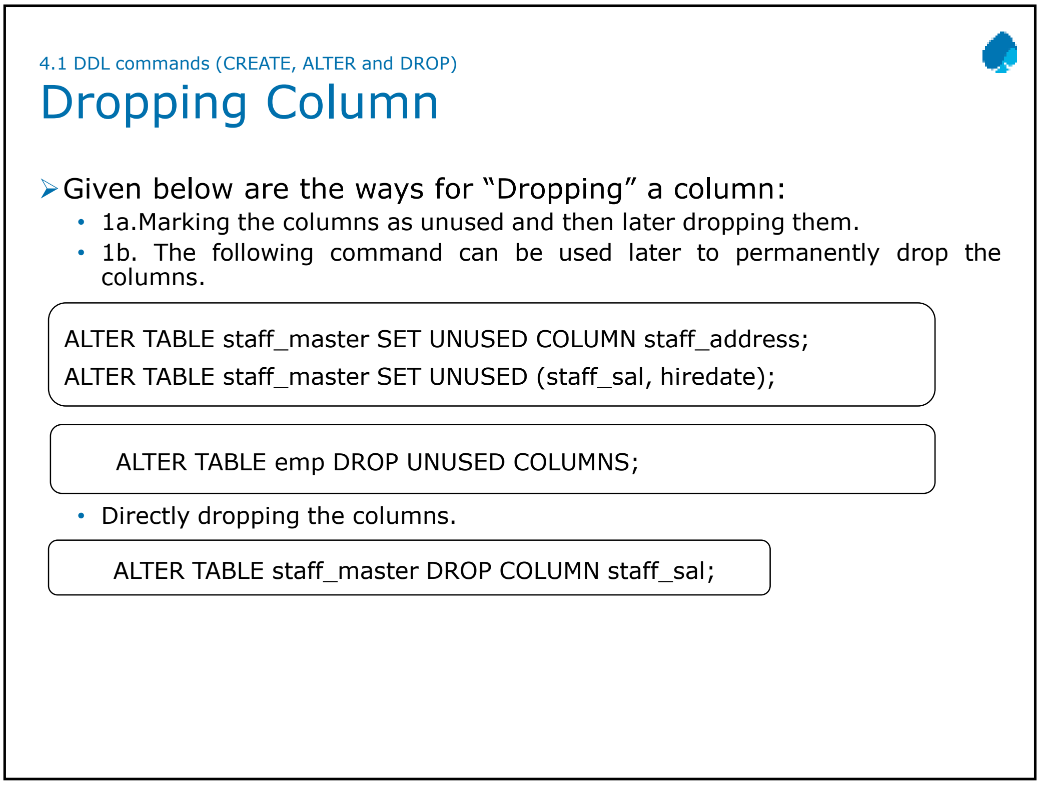
```
ALTER TABLE  student_master
DROP CONSTRAINT  stu_dept_fk ;
```

ALTER TABLE – Drop Clause

- To remove the PRIMARY KEY constraint on the DEPARTMENT table and drop the associated FOREIGN KEY constraint on the DEPT_CODE column.

```
ALTER TABLE department_master
DROP PRIMARY KEY CASCADE;
```

Instructor Notes:



4.1 DDL commands (CREATE, ALTER and DROP)

Dropping Column

- Given below are the ways for "Dropping" a column:
  - 1a.Marking the columns as unused and then later dropping them.
  - 1b. The following command can be used later to permanently drop the columns.

```
ALTER TABLE staff_master SET UNUSED COLUMN staff_address;
ALTER TABLE staff_master SET UNUSED (staff_sal, hiredate);
```

```
ALTER TABLE emp DROP UNUSED COLUMNS;
```

  - Directly dropping the columns.

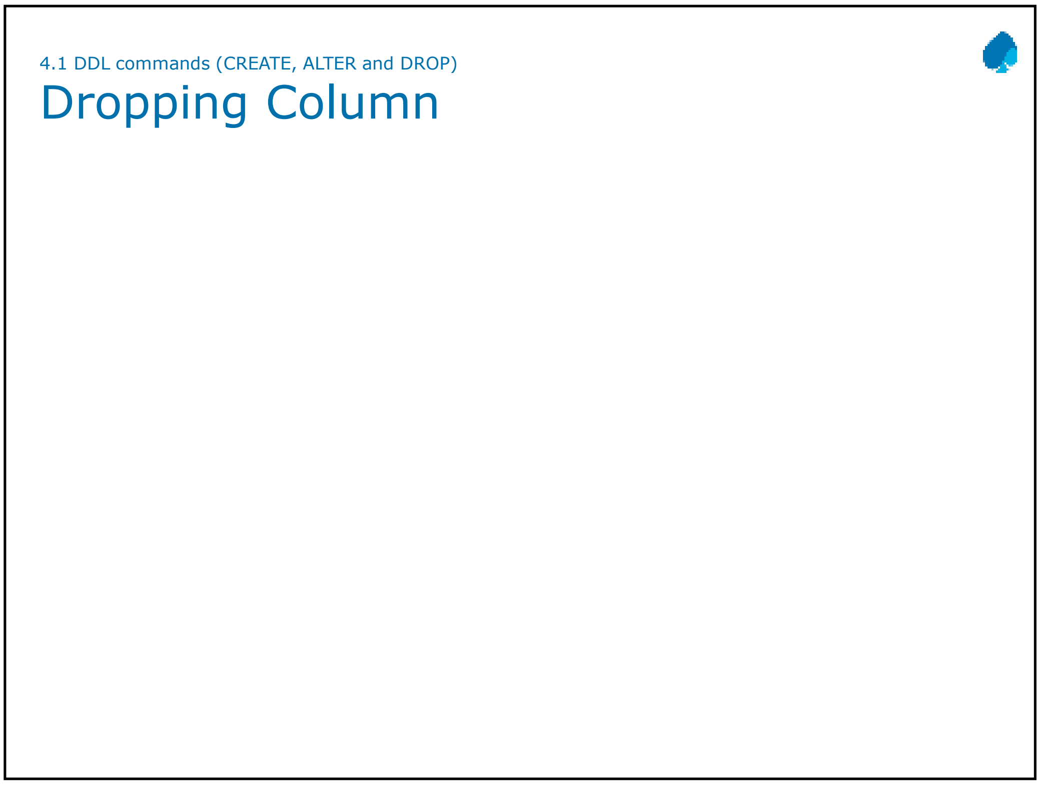
```
ALTER TABLE staff_master DROP COLUMN staff_sal;
```

<u>**Ways for "Dropping" a column:**</u>

1. **Marking the columns as "Unused" and then later dropping them:**

- Oracle onwards a new feature to "drop" and "set" the "unused columns" in a table is added
  - First command as shown in the slide, allows you to mark a column as unused and
  - Second command as shown in the following slide, lets you drop the unused column from the table to create more free space.
    - This feature drops the column from a table and releases any space back to the segment.
- **Note that:**
  - Columns once marked as unused cannot be recovered.
  - Marking the columns as unused does not release the space occupied by them back to the database.
    - Until you actually drop these columns, they continue to count towards the absolute limit of 1000 columns per table.
    - If you mark a column of data type LONG as UNUSED, you cannot add another LONG column to the table until you actually drop the unused LONG column.
- The advantage of the marking column as "unused" and then dropping them is that marking the columns is much faster process than dropping the columns.
- You can refer to the data dictionary table USER_UNUSED_COL_TABS to get information regarding the tables with columns marked as unused.

Instructor Notes:



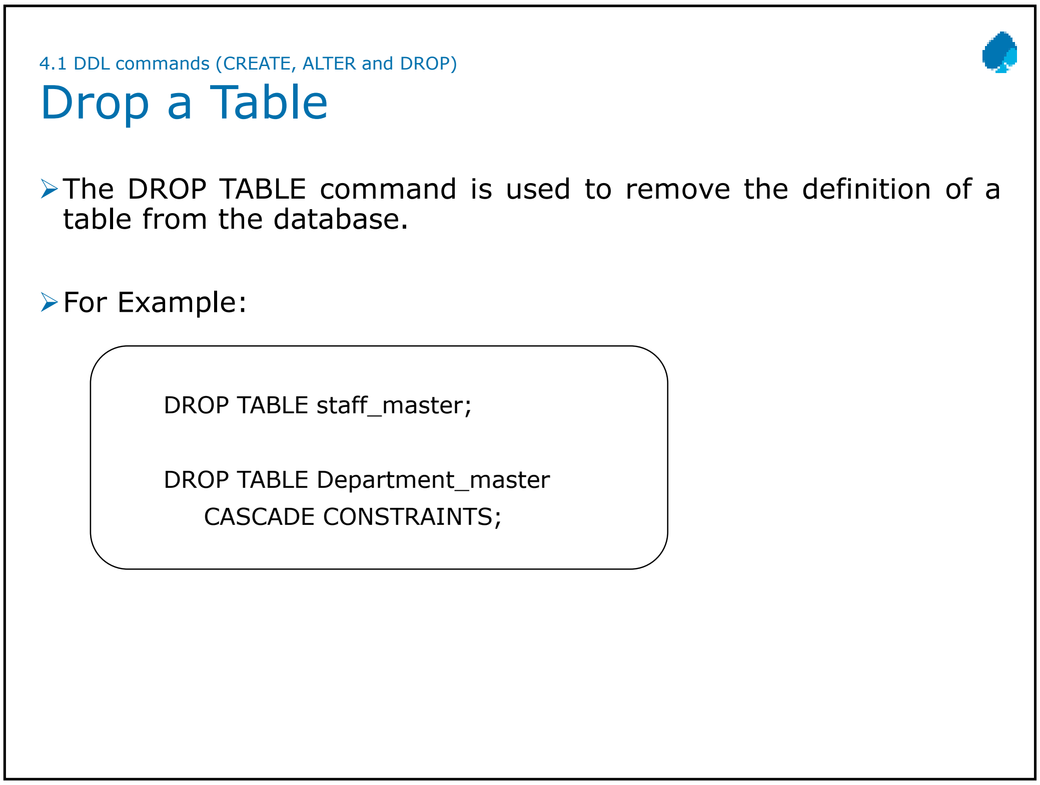
Dropping Column

Ways for "Dropping" a column (contd.):

2. **DROP COLUMN:**

- This feature allows directly dropping the column.
- For DROP COLUMN command shown in the slide, to work successfully, the table should be exclusively locked by the user by giving the command.
  - ➢ All "indexes" defined on any of the target columns are also dropped.
  - ➢ All "constraints" that reference a target column are removed.
- Note that this command should be used with caution.
- The CASCADE CONSTRAINTS clause is used along with the DROP COLUMN clause.
- The CASCADE CONSTRAINTS clause drops all referential integrity constraints that refer to the primary and unique keys defined on the dropped columns.
- The CASCADE CONSTRAINTS clause also drops all multicolumn constraints defined on the dropped columns

Instructor Notes:



4.1 DDL commands (CREATE, ALTER and DROP)

Drop a Table

- The DROP TABLE command is used to remove the definition of a table from the database.

- For Example:

```
DROP TABLE staff_master;

DROP TABLE Department_master
    CASCADE CONSTRAINTS;
```

Deleting Database Objects: Tables

- Deleting objects that exist in the database is an easy task. Just say:

```
DROP Obj_Type obj_name;
```

- A table that is dropped cannot be recovered. When a table is dropped, dependent objects such as indexes are automatically dropped. Synonyms and views created on the table remain, but give an error if they are referenced.
- You cannot delete a table that is being referenced by another table. To do so use the following:

```
DROP  TABLE table-name CASCADE CONSTRAINTS;
```

Instructor Notes:

What is Data Integrity?

- Data Integrity:
  - "Data Integrity" allows to define certain "data quality requirements" that must be met by the data in the database.
  - Oracle uses "Integrity Constraints" to prevent invalid data entry into the base tables of the database.
    - You can define "Integrity Constraints" to enforce the business rules you want to associate with the information in a database.
    - If any of the results of a "DML statement" execution violate an "integrity constraint", Oracle rolls back the statement and returns an error.

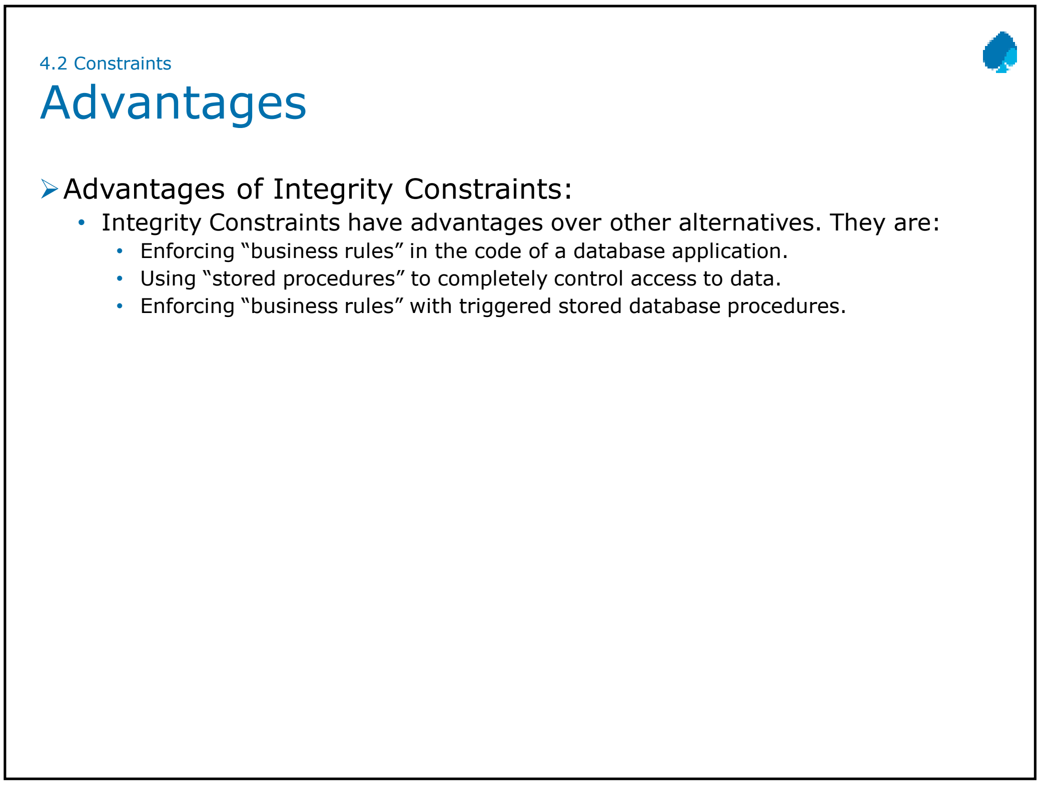
Data Integrity: Integrity Constraint

- An Integrity Constraint is a declarative method of defining a rule for a column of a table.

Example of Data Integrity:

- Assume that you define an "Integrity Constraint" for the Staff_Sal column of the Staff_Master table.
- This Integrity Constraint enforces the rule that "no row in this table can contain a numeric value greater than 10,000 in this column".
- If an INSERT or UPDATE statement attempts to violate this "Integrity Constraint", Oracle rolls back the statement and returns an "information error" message.

Instructor Notes:

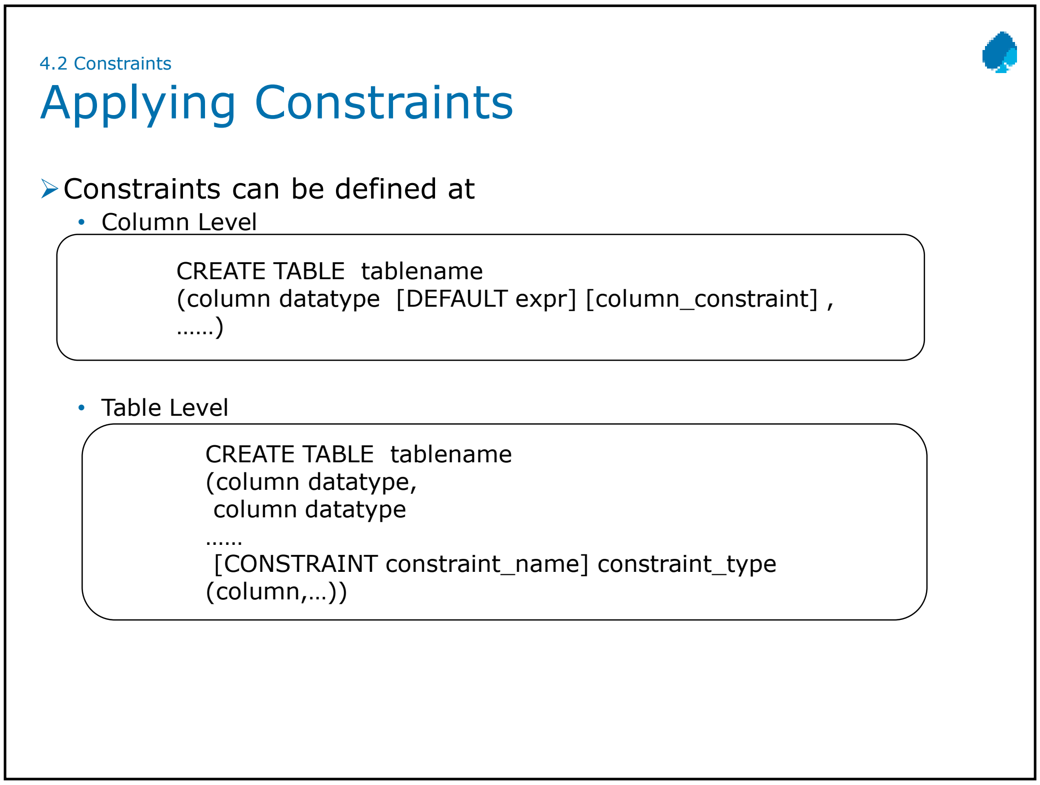


4.2 Constraints

Advantages

- Advantages of Integrity Constraints:
  - Integrity Constraints have advantages over other alternatives. They are:
    - Enforcing "business rules" in the code of a database application.
    - Using "stored procedures" to completely control access to data.
    - Enforcing "business rules" with triggered stored database procedures.

Instructor Notes:



4.2 Constraints

Applying Constraints

- Constraints can be defined at
  - Column Level

```
CREATE TABLE  tablename
(column datatype  [DEFAULT expr] [column_constraint] ,
……)
```

  - Table Level

```
CREATE TABLE  tablename
(column datatype,
 column datatype
……
 [CONSTRAINT constraint_name] constraint_type
(column,…))
```

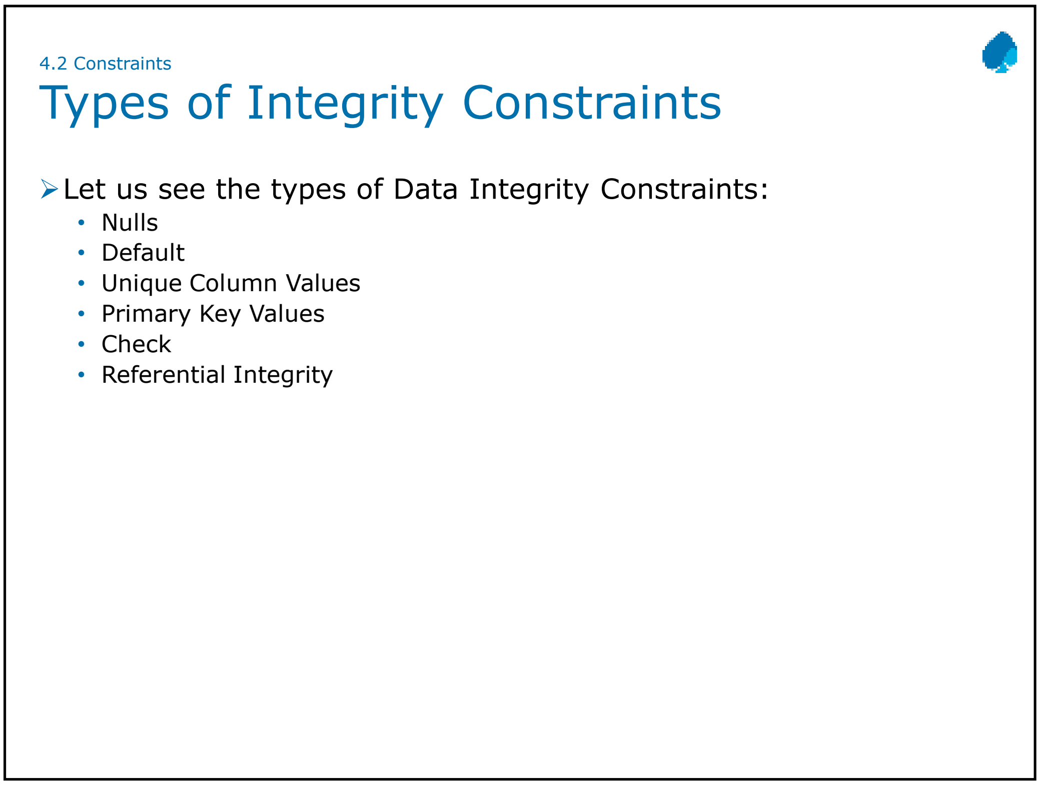
<u>Applying Constraints:</u>

In Oracle you can apply constraints

Column Level - References a single column and is defined within a specification for the owning column; can be any type of integrity constraint

Table Level - References one or more columns and is defined separately from the definitions of the columns in the table; can define any constraints except NOT NULL

Instructor Notes:



4.2 Constraints

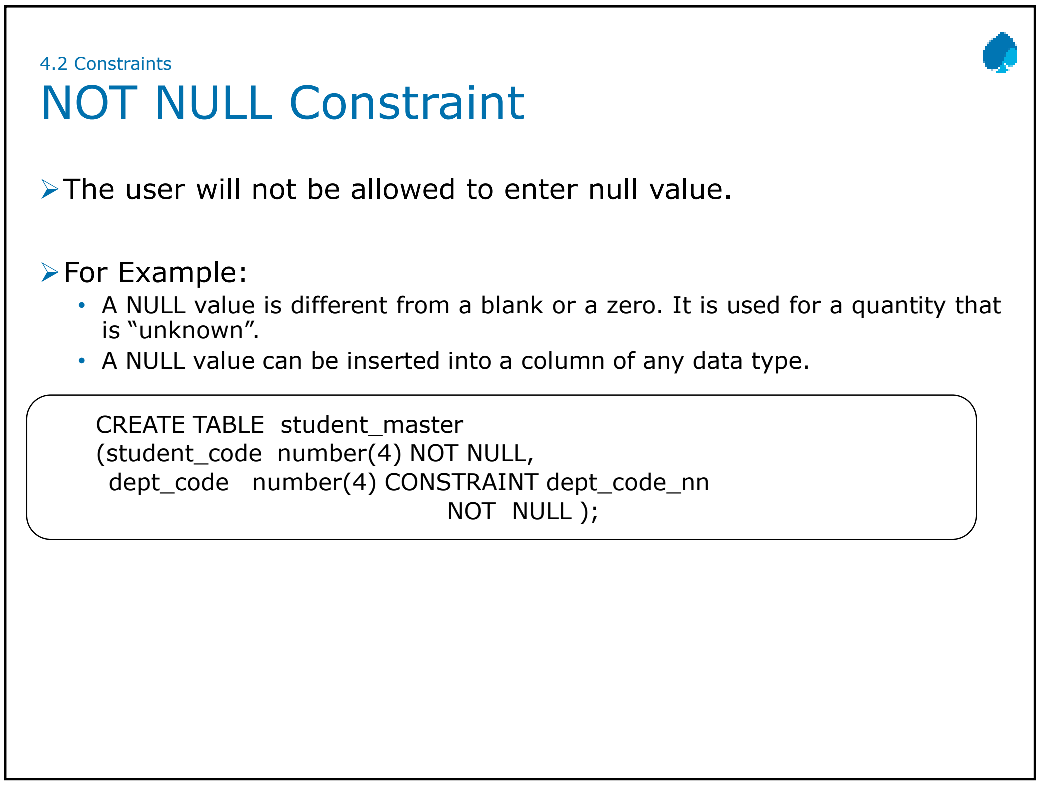
Types of Integrity Constraints

- Let us see the types of Data Integrity Constraints:
  - Nulls
  - Default
  - Unique Column Values
  - Primary Key Values
  - Check
  - Referential Integrity

<u>Types of Data Integrity:</u>

- Oracle supports the following Integrity Constraints:
  - NOT NULL constraints for the rules associated with nulls in a column. "Null" is a rule defined on a single column that allows or disallows, inserts or updates on rows containing a "null" value (the absence of a value) in that column.
  - UNIQUE key constraints for the rule associated with unique column values. A "unique value" rule defined on a column (or set of columns) allows inserting or updating a row only if it contains a "unique value" in that column (or set of columns).
  - PRIMARY KEY constraints for the rule associated with primary identification values. A "primary key" value rule defined on a key (a column or set of columns) specifies that "each row in the table can be uniquely identified by the values in the key".
  - FOREIGN KEY constraints for the rules associated with referential integrity. A "Referential Integrity" rule defined on a key (a column or set of columns) in one table guarantees that "the values in that key, match the values in a key in a related table (the referenced value)". Oracle currently supports the use of FOREIGN KEY integrity constraints to define the referential integrity actions, including:
    - update and delete No Action
    - delete CASCADE
    - delete SET NULL
  - CHECK constraints for complex integrity rules

Instructor Notes:



4.2 Constraints

NOT NULL Constraint

- The user will not be allowed to enter null value.

- For Example:
  - A NULL value is different from a blank or a zero. It is used for a quantity that is "unknown".
  - A NULL value can be inserted into a column of any data type.

```
CREATE TABLE  student_master
(student_code  number(4) NOT NULL,
 dept_code   number(4) CONSTRAINT dept_code_nn
                               NOT  NULL );
```

<u>NOT NULL constraint:</u>

- Often there may be records in a table that do not have values for every field.
  - This could be because the information is not available at the time of the data entry or because the field is not applicable in every case.
- If the column is created as NULLABLE, in the absence of a user-defined value the DBMS will place a NULL value in the column.
- A NULL value is different from a blank or a zero. It is used for a quantity that is "unknown".
- "Null" is a rule defined on a single column that allows or disallows, inserts or updates on rows containing a "null" value (the absence of a value) in that column.
- A NULL value can be inserted into a column of any data type.
- **Principles of NULL values:**
  - Setting a NULL value is appropriate when the "actual value" is unknown, or when a value is not meaningful.
  - A NULL value is not equivalent to the value of "zero" if the data type is number, and it is not equivalent to "spaces" if the data type is character.
  - A NULL value will evaluate to NULL in any expression
    For example: NULL multiplied by 10 is NULL.
  - NULL value can be inserted into columns of any data type.
  - If the column has a NULL value, Oracle ignores any UNIQUE, FOREIGN KEY, and CHECK constraints that may be attached to the column.

Instructor Notes:

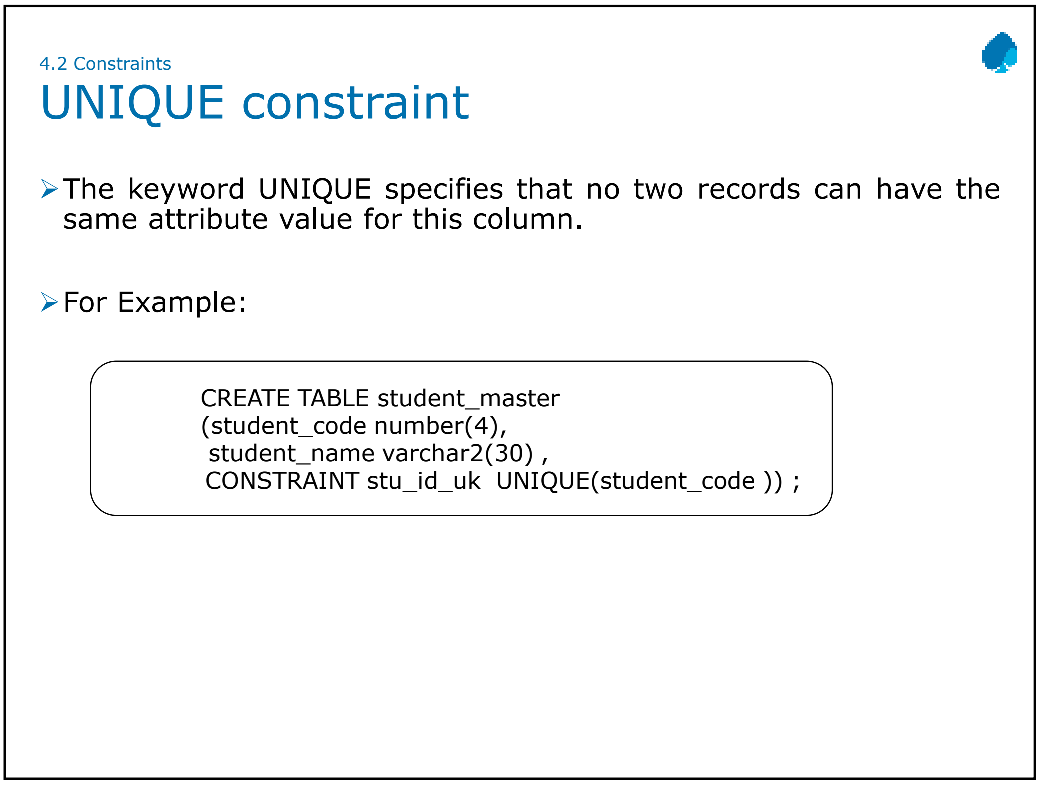
DEFAULT clause

- If no value is given, then instead of using a “Not Null” constraint, it is sometimes useful to specify a default value for an attribute.

- For Example:
  - When a record is inserted the default value can be considered.

```
CREATE TABLE  staff_master(
Staff_Code number(8) PRIMARY KEY,
Staff_Name varchar2(50) NOT NULL,
Staff_dob date,
Hiredate date DEFAULT sysdate,
…..)
```

Instructor Notes:



4.2 Constraints

UNIQUE constraint

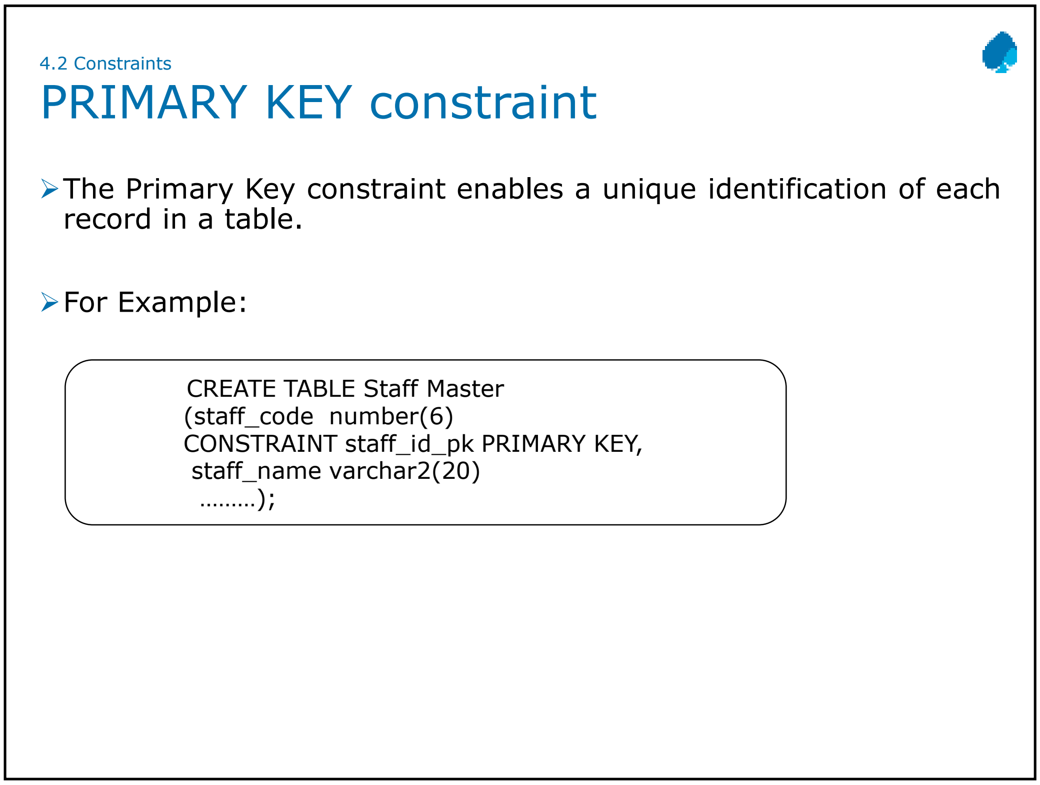
- The keyword UNIQUE specifies that no two records can have the same attribute value for this column.
- For Example:

```
CREATE TABLE student_master
(student_code number(4),
 student_name varchar2(30) ,
 CONSTRAINT stu_id_uk  UNIQUE(student_code )) ;
```

UNIQUE constraint:

- The UNIQUE constraint does not allow duplicate values in a column.
  - If the UNIQUE constraint encompasses two or more columns, then two equal combinations are not allowed.
  - However, if a column is not explicitly defined as NOT NULL, then NULLS can be inserted multiple number of times.
- A UNIQUE constraint can be extended over multiple columns.
- A “unique value” rule defined on a column (or set of columns) allows inserting or updating a row only if it contains a “unique value” in that column (or set of columns).

Instructor Notes:



4.2 Constraints

PRIMARY KEY constraint

- The Primary Key constraint enables a unique identification of each record in a table.
- For Example:

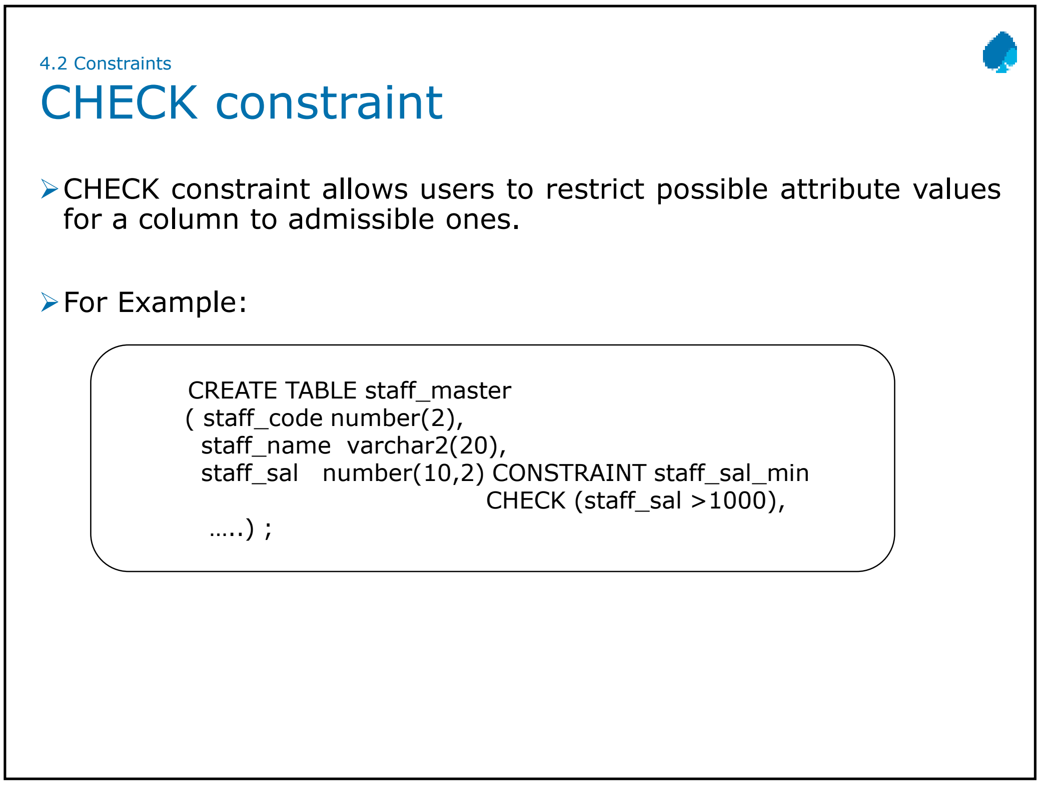
```
CREATE TABLE Staff Master
(staff_code  number(6)
CONSTRAINT staff_id_pk PRIMARY KEY,
 staff_name varchar2(20)
  .........);
```

PRIMARY KEY constraint:

- On a technical level, a PRIMARY KEY combines a UNIQUE constraint and a NOT NULL constraint.
- Additionally, a table can have at the most one PRIMARY KEY.
- After creating a PRIMARY KEY, it can be referenced by a FOREIGN KEY.
- A “primary key” value rule defined on a key (a column or set of columns) specifies that “each row in the table can be uniquely identified by the values in the key”.
- The example on the slide defines the primary key constraint at the column level. The same example is seen below with the constraint defined at table level

```
          CREATE TABLE Staff Master
          (staff_code  number(6) ,
           staff_name varchar2(20),
            .........
         CONSTRAINT staff_id_pk  PRIMARY KEY
      (staff_code))
      ;
```

Instructor Notes:



CHECK constraint

- CHECK constraint allows users to restrict possible attribute values for a column to admissible ones.
- For Example:

```
CREATE TABLE staff_master
( staff_code number(2),
  staff_name  varchar2(20),
  staff_sal   number(10,2) CONSTRAINT staff_sal_min
                                       CHECK (staff_sal >1000),
  …..) ;
```

<u>CHECK constraint:</u>

- A CHECK constraint allows to state a minimum requirement for the value in a column.
- If more complicated requirements are desired, an INSERT trigger must be used.

Instructor Notes:

FOREIGN KEY constraint

- The FOREIGN KEY constraint specifies a "column" or a "list of columns" as a foreign key of the referencing table.
- The referencing table is called the "child-table", and the referenced table is called "parent-table".
- For Example:

```
CREATE TABLE student_master
(student_code number(6) ,
 dept_code number(4) CONSTRAINT stu_dept_fk
                REFERENCES department_master(dept_code),
 student_name varchar2(30) );
```

FOREIGN KEY Constraint Keywords:

- Given below are a few 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.
- You can query the USER_CONSTRAINTS table to view all constraint definitions and names.
- You can view the columns associated with the constraint names in the USER_CONS_COLUMNS view.

- In EMP table, for deptno column, if we want to allow only those values that already exist in deptno column of the DEPT table, we must enforce what is known as REFERENTIAL INTEGRITY. To enforce REFERENTIAL INTEGRITY, declare deptno field of DEPT table as PRIMARY KEY, and deptno field of EMP table as FOREIGN KEY as follows

Instructor Notes:

FOREIGN KEY Constraint Keywords:

- In the given example, FOREIGN KEY has been declared as a Table constraint.

```
CREATE TABLE Dept
(
Deptno       NUMBER(2) CONSTRAINT
DEPTNO_P_KEY PRIMARY KEY,
Dname        VARCHAR2(14)     NOT NULL,
Loc          VARCHAR2(13)     NOT NULL
     );
CREATE TABLE Emp
     (
     Empno NUMBER(4) CONSTRAINT P_KEY
PRIMARY KEY,
     Ename VARCHAR2(10)  CONSTRAINT
ENAME_NOT_NULL  NOT NULL,
     Deptno NUMBER(2),
Job CHAR(9) CONSTRAINT JOB_ALL_UPPER
CHECK (Job =UPPER(Job)),
     Hiredate DATE DEFAULT SYSDATE,
     CONSTRAINT DEPTNO_F_KEY FOREIGN KEY
(Deptno)
   REFERENCES Dept(Deptno)
     );
```

Instructor Notes:

Creation of database objects: Tables (contd.):

- A table can have a maximum of 1000 columns. Only one column of type LONG is allowed per table.

Table_name, col_name, const_name	A string upto 30 characters length. Can be made up to A-Z,0-9,$,_,# Must begin with a non-numeric ORACLE data characters.
Col_datatype	One of the previously mentioned types.
Col_constraint	A restriction on the column can be of following types: PRIMARY KEY, NOT NULL, UNIQUE, FOREIGN KEY, CHECK, Can be named. Only one PRIMARY KEY is allowed per table.
Table_constraint	A restriction on single or multiple columns. The types are same as in col_constraint. (NULL constraint is not allowed here).
AS query	Query is an SQL statement using SELECT command. SELECT command returns the rows from tables. It is useful if the table being created is based on an existing table. New table need not have all the columns of the old table. Names of columns can be different. All or part of the data can be copied.

- Any object created by a user is accessible to the user and the DBA only.
- To make the object accessible to other users, the creator or the DBA must explicitly give permission to others.

Instructor Notes:

4.3 Sequence

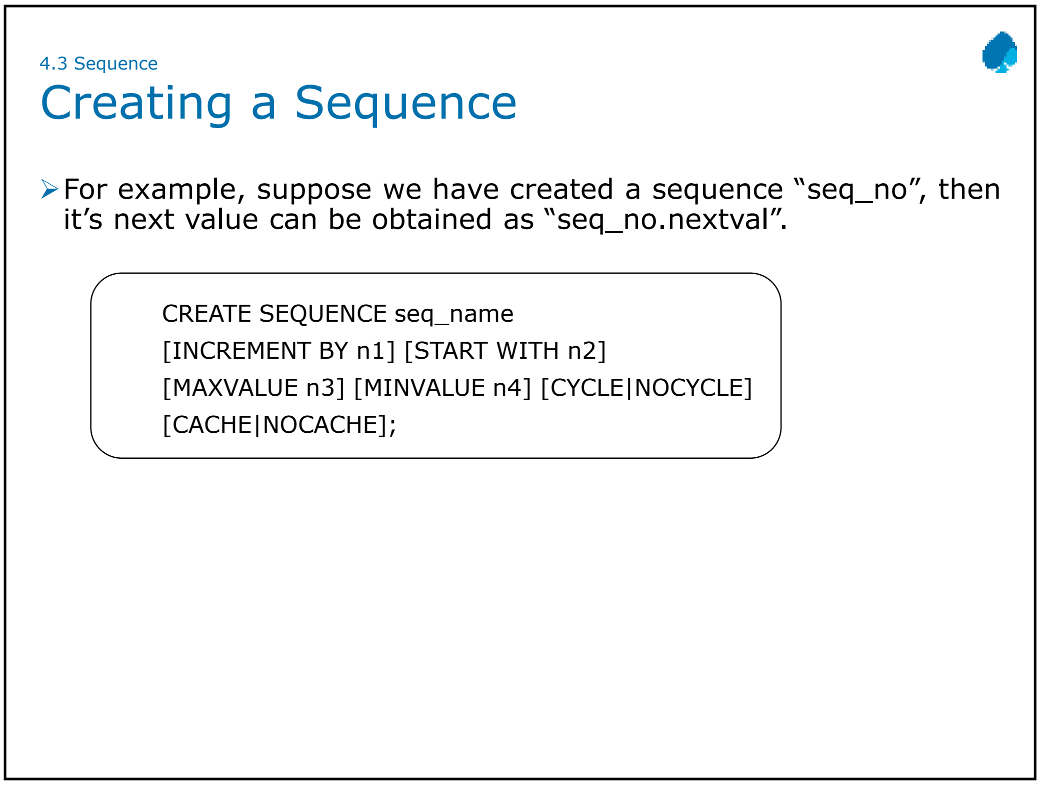
Usage of Sequence

- A "Sequence" is an object, which can be used to generate sequential numbers.
- A Sequence is used to fill up columns, which are declared as UNIQUE or PRIMARY KEY.
- A Sequence uses "NEXTVAL" to retrieve the next value in the sequence order.

Sequence:

- A Sequence:
  - automatically generates unique numbers.
  - is a "sharable object".
  - is typically used to create a PRIMARY KEY value.
  - replaces application code.
  - speeds up the efficiency of accessing sequence values when cached in memory.

Instructor Notes:



4.3 Sequence

Creating a Sequence

- For example, suppose we have created a sequence “seq_no”, then it’s next value can be obtained as “seq_no.nextval”.

```
CREATE SEQUENCE seq_name
[INCREMENT BY n1] [START WITH n2]
[MAXVALUE n3] [MINVALUE n4] [CYCLE|NOCYCLE]
[CACHE|NOCACHE];
```

<u>Sequence</u>:

- In the example shown in the slide:
  - START WITH indicates the first NUMBER in the series.
  - INCREMENT BY is the difference between consecutive numbers. If n1 is negative, then the numbers that are generated are in a descending order.
  - MAXVALUE and MINVALUE indicate the extreme values.
  - CYCLE indicates that once the extreme is reached, it starts the cycle again with n2.
  - NOCYCLE means that once the extreme is reached, it stops generating numbers.
  - CACHE caches the specified number of sequence values in the memory. This speeds access, but all cached numbers are lost when the database is shut down. The default value is 20

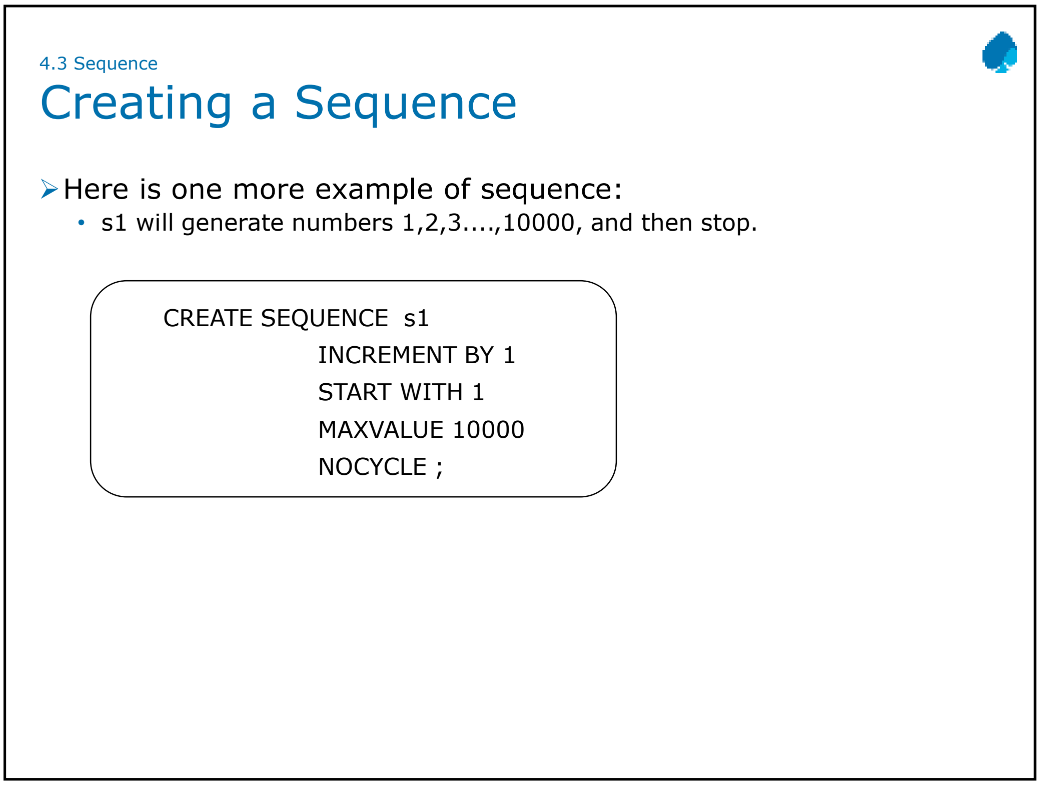
Confirming Sequences

- As shown below, you need to verify your sequence values in the USER_SEQUENCES data dictionary table.

```
SELECT sequence_name, min_value, max_value,
         increment_by, last_number
FROM user_sequences;
```

- The LAST_NUMBER column displays the next available sequence number if NOCACHE is specified.

Instructor Notes:



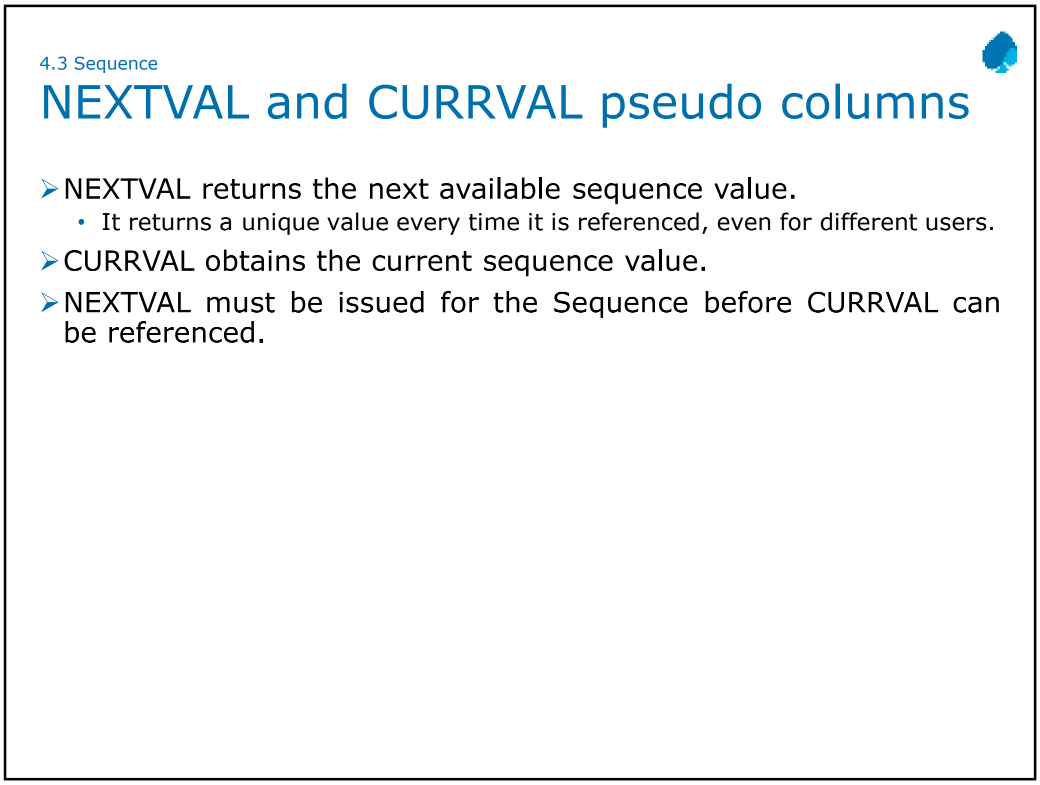
4.3 Sequence

Creating a Sequence

- Here is one more example of sequence:
  - s1 will generate numbers 1,2,3….,10000, and then stop.

```
CREATE SEQUENCE  s1
               INCREMENT BY 1
               START WITH 1
               MAXVALUE 10000
               NOCYCLE ;
```

Instructor Notes:



NEXTVAL and CURRVAL pseudo columns

- NEXTVAL returns the next available sequence value.
  - It returns a unique value every time it is referenced, even for different users.
- CURRVAL obtains the current sequence value.
- NEXTVAL must be issued for the Sequence before CURRVAL can be referenced.

Referencing a Sequence:

- After you create a Sequence, it generates sequential numbers that can be used in your tables. You can reference the Sequence values by using the NEXTVAL and CURRVAL pseudocolumns.
- **NEXTVAL and CURRVAL Pseudocolumns:**
  - The **NEXTVAL pseudocolumn** is used to extract successive sequence numbers from a specified Sequence. You must qualify NEXTVAL with the sequence name. When you reference sequence.NEXTVAL, a new sequence number is generated and the current sequence number is placed in CURRVAL.
  - The **CURRVAL pseudocolumn** is used to refer a Sequence number that the current user has just generated.
- NEXTVAL must be used to generate a sequence number in the session of the current user, before CURRVAL can be referenced.
  - You must qualify CURRVAL with the sequence name.
  - When “sequence.CURRVAL” is referenced, the last value returned to that user’s process is displayed.

Instructor Notes:

Drop a Sequence

- A Sequence can be removed from the data dictionary by using the DROP SEQUENCE statement.
- Once removed, the Sequence can no longer be referenced.

```
DROP SEQUENCE dept_deptid_seq;
Sequence dropped.
```

Removing a Sequence

- To remove a sequence from the data dictionary, use the DROP SEQUENCE statement.
- To remove the Sequence, you must be the owner of the Sequence or possess the DROP ANY SEQUENCE privilege.

Syntax:

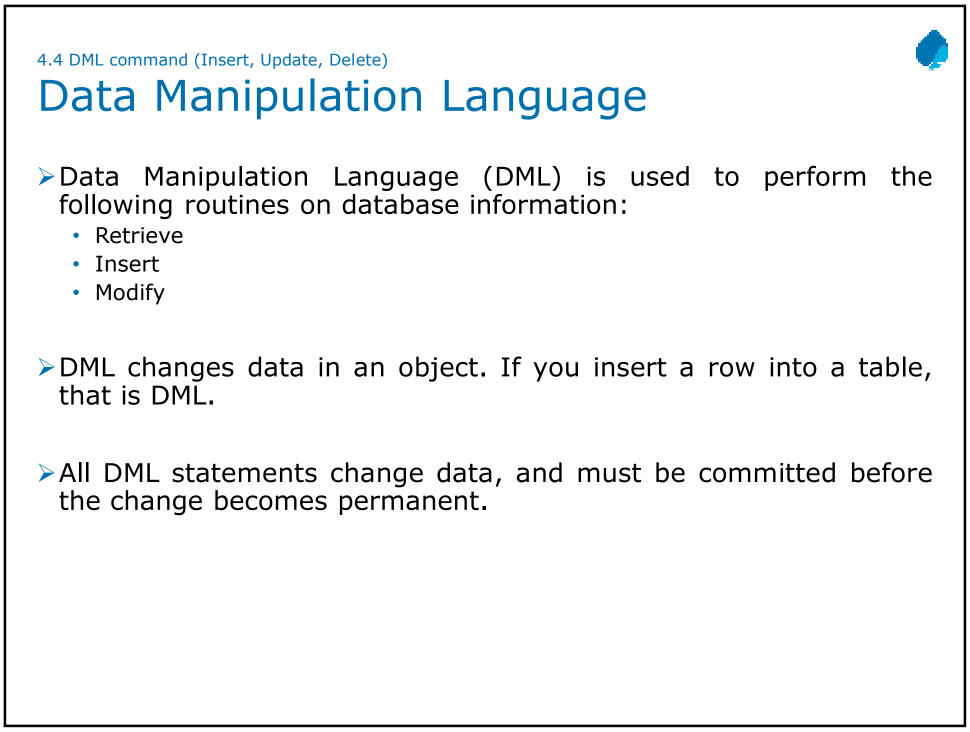
```
DROP   SEQUENCE    sequence;
```

where: sequence is the name of the sequence generator.

- For more information, refer Oracle9i SQL Reference, “DROP SEQUENCE”.

contd.

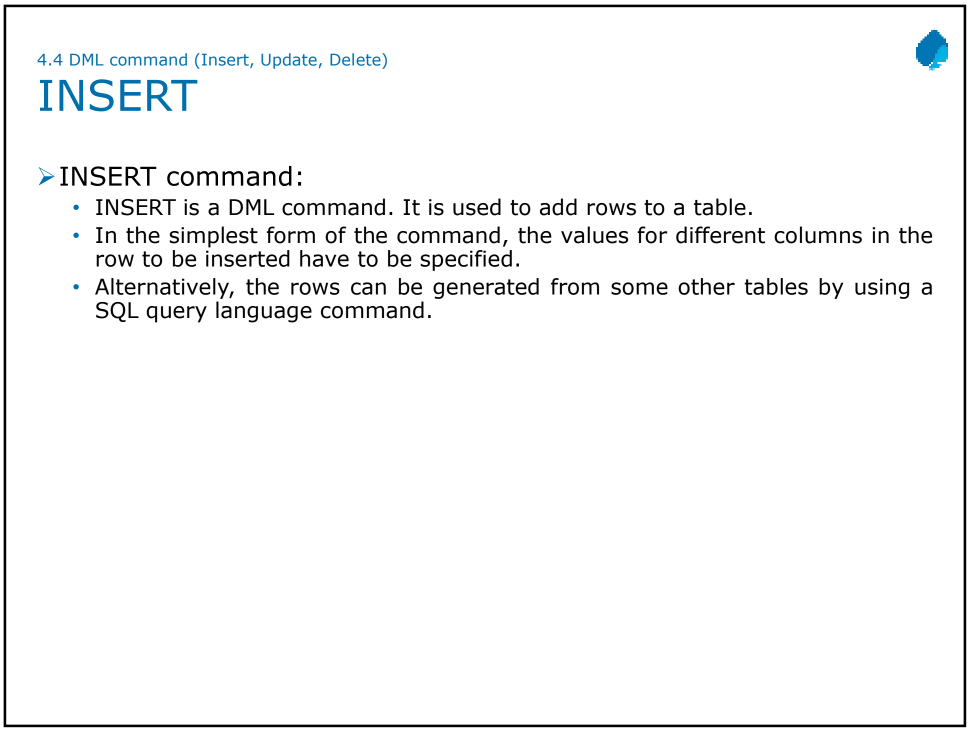
Instructor Notes:



Data Manipulation Language

- Data Manipulation Language (DML) is used to perform the following routines on database information:
  - Retrieve
  - Insert
  - Modify
- DML changes data in an object. If you insert a row into a table, that is DML.
- All DML statements change data, and must be committed before the change becomes permanent.

Instructor Notes:



4.4 DML command (Insert, Update, Delete)

INSERT

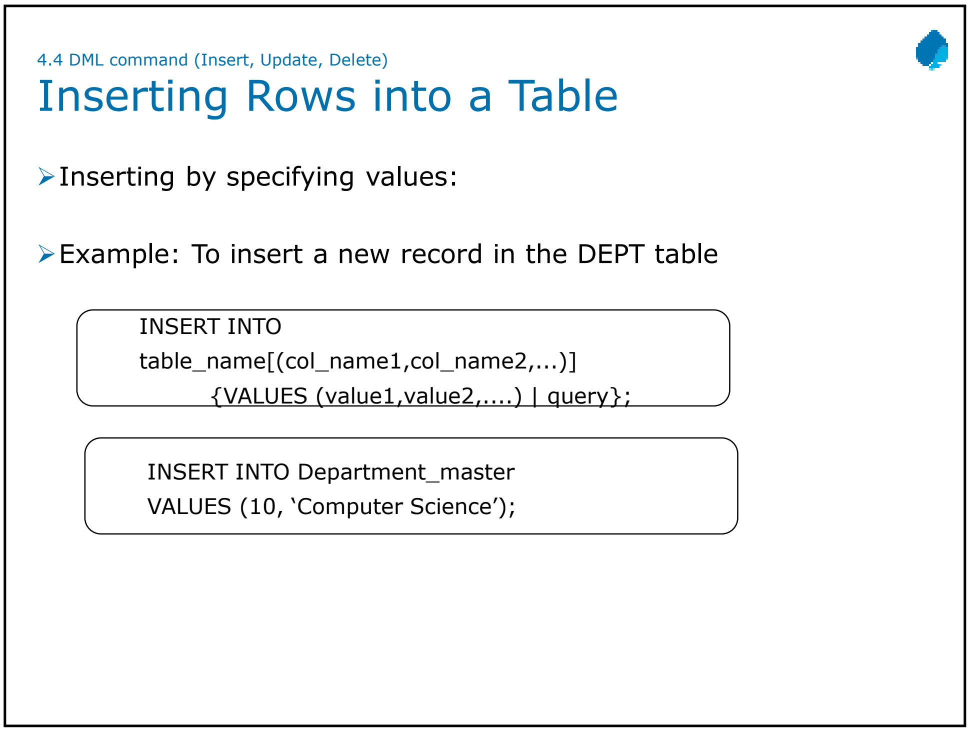
- INSERT command:
  - INSERT is a DML command. It is used to add rows to a table.
  - In the simplest form of the command, the values for different columns in the row to be inserted have to be specified.
  - Alternatively, the rows can be generated from some other tables by using a SQL query language command.

<u>Addition of Data into Tables:</u>

Requisites for using INSERT command:

- If values are specified for all columns in the order specified at creation, then col_names could be omitted.
- Values should match "data type" of the respective columns.
- Number of values should match the number of column names mentioned.
- All columns declared as NOT NULL should be supplied with a value.
- Character strings should be enclosed in quotes.
- Date values should be enclosed in quotes.
- Values will insert one row at a time.
- Query will insert all the rows returned by the query.
- The table_name can be a "table" or a "view". If table_name is a "view", then the following restrictions apply:
  1. The "view" cannot have a GROUP BY, CONNECT BY, START WITH, DISTINCT, UNION, INTERSECT, or MINUS clause or a join.
  2. If the "view" has WITH CHECK OPTION clause, then a row, which will not be returned by the view, cannot be inserted.

Instructor Notes:



4.4 DML command (Insert, Update, Delete)

Inserting Rows into a Table

- Inserting by specifying values:
- Example: To insert a new record in the DEPT table

```
INSERT INTO
table_name[(col_name1,col_name2,...)]
        {VALUES (value1,value2,....) | query};
```

```
INSERT INTO Department_master
VALUES (10, ‘Computer Science’);
```

Inserting Rows into a Table:

Example:

- Inserting a row in EMP table giving all values.

```
INSERT INTO student_master
VALUES(1001,'Amit',10,'11-Jan-80',‘Chennai');
```

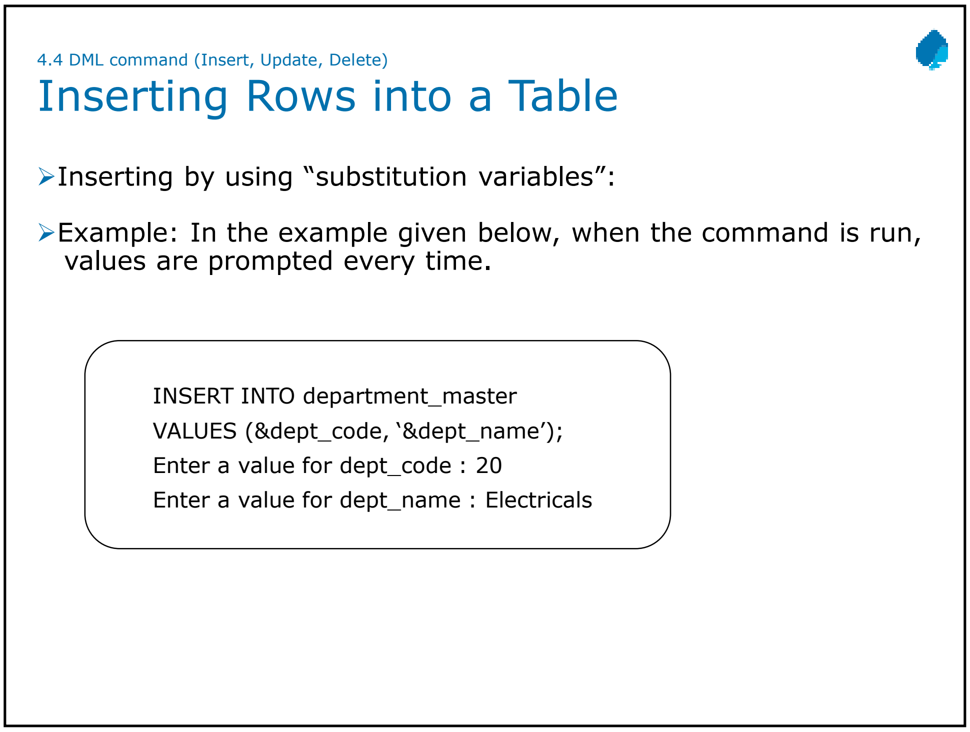
- 10 is a dept number which exists in DEPARTMENT_MASTER table

- Inserting a row in STAFF_MASTER table giving some values.

```
INSERT INTO staff_master
(staff_code,staff_name,design_code,dept_code)
VALUES(100001,'Arvind',102,30);
```

- This row will be created if all the constraints like NOT NULL are satisfied.

Instructor Notes:



4.4 DML command (Insert, Update, Delete)

Inserting Rows into a Table

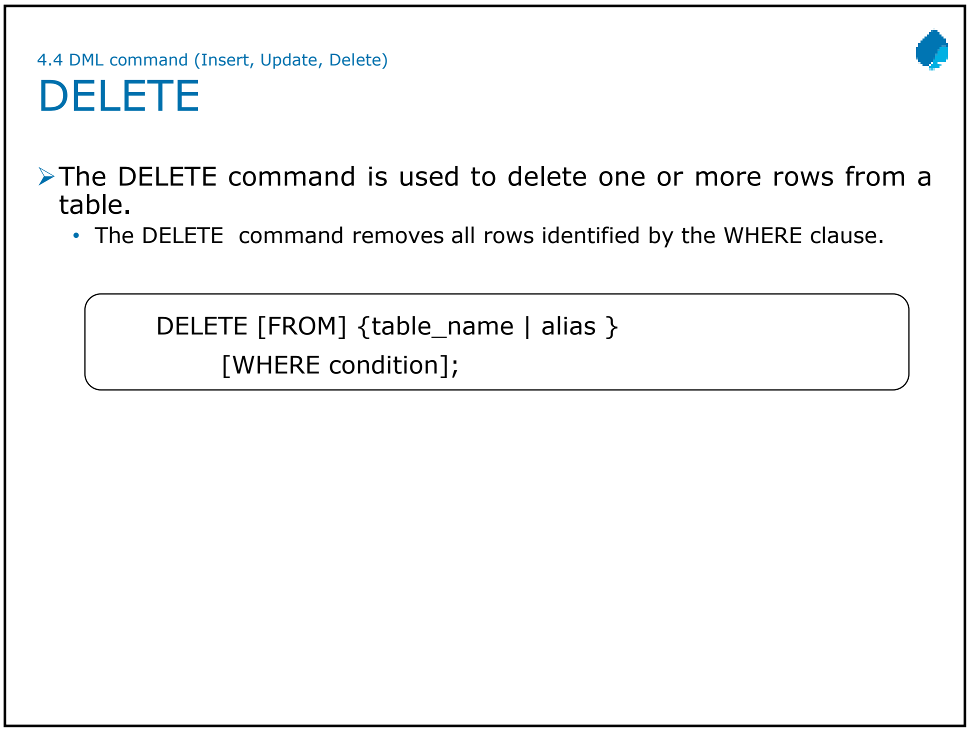
- Inserting by using “substitution variables”:
- Example: In the example given below, when the command is run, values are prompted every time.

```
INSERT INTO department_master
VALUES (&dept_code, ‘&dept_name’);
Enter a value for dept_code : 20
Enter a value for dept_name : Electricals
```

Inserting Rows into a Table:
Inserting by using “substitution variables”:

- The problem with the INSERT statement is that it adds only “one row” to the table.
- However, by using “substitution variables” the speed of data input can be increased.
- Whenever a “substitution variable” is placed in a “value” field, the user will be prompted to enter the “actual value” when the command is executed.

Instructor Notes:



4.4 DML command (Insert, Update, Delete)

DELETE

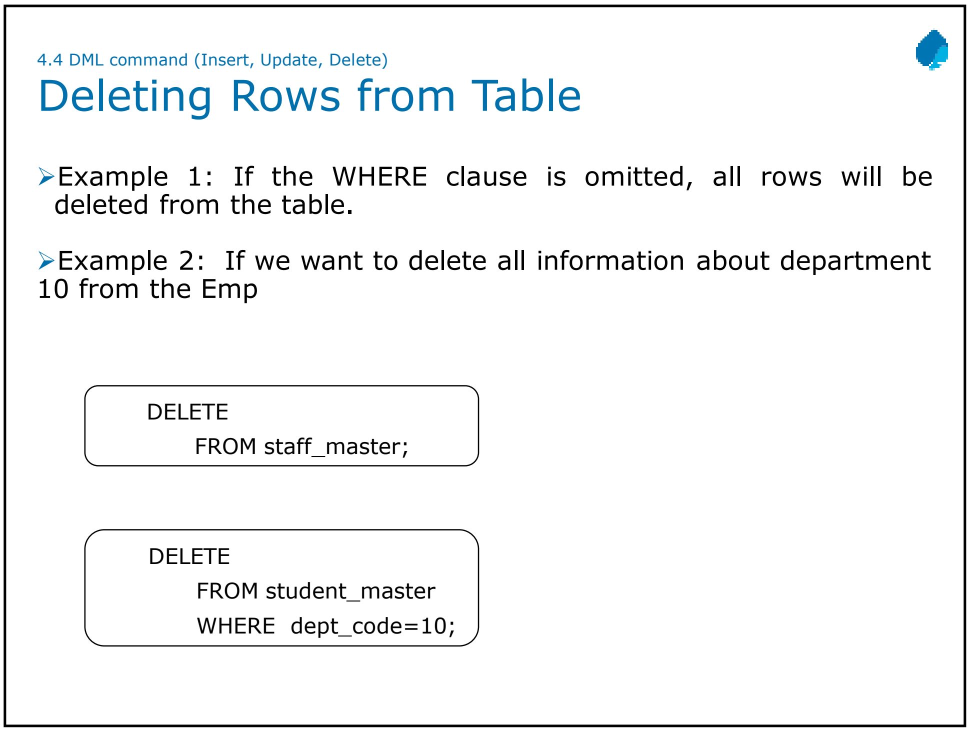
- The DELETE command is used to delete one or more rows from a table.
  - The DELETE command removes all rows identified by the WHERE clause.

```
DELETE [FROM] {table_name | alias }
      [WHERE condition];
```

Deletion of Data from Tables

- The table_name can be a “table” or a “view”.
- The DELETE command is used to delete one or more rows from a table.
- The DELETE statement removes all rows identified by the WHERE clause.
  - This is another DML, which means we can rollback the deleted data, and that to make our changes permanent.
- If WHERE clause is omitted, all rows from the table are removed. Else all rows which satisfy the condition are removed.
- FROM clause can be omitted without affecting the statement.

Instructor Notes:



4.4 DML command (Insert, Update, Delete)

Deleting Rows from Table

- Example 1: If the WHERE clause is omitted, all rows will be deleted from the table.

- Example 2: If we want to delete all information about department 10 from the Emp

```
DELETE
    FROM staff_master;
```

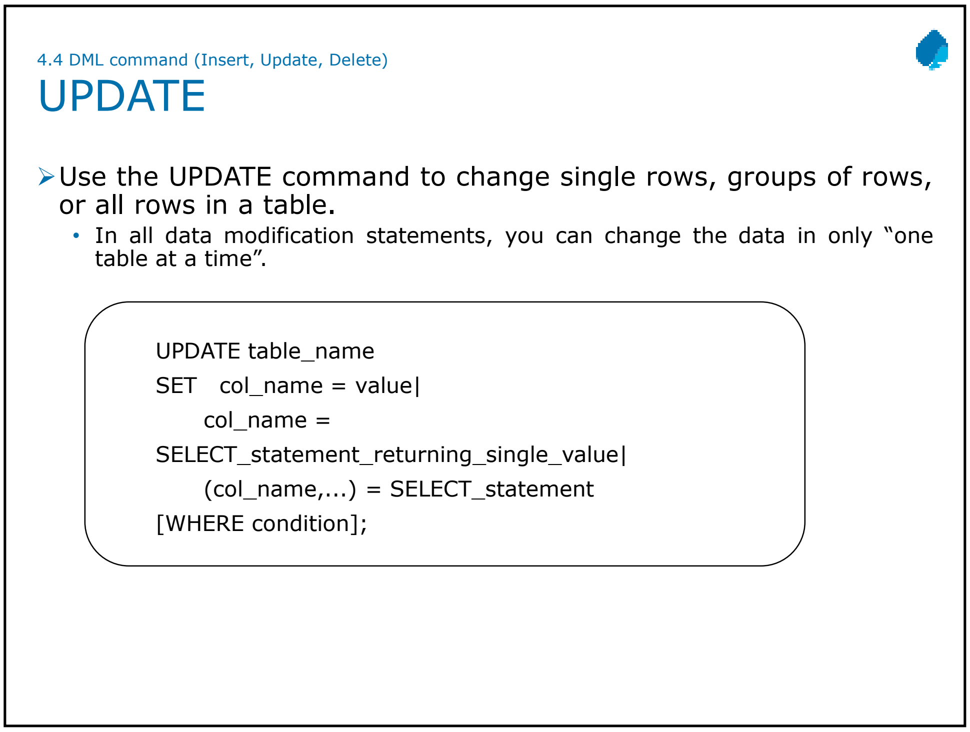
```
DELETE
    FROM student_master
    WHERE  dept_code=10;
```

<u>Deletion of Data from Tables</u>

Example 3:

```
DELETE staff_master WHERE staff_name = ‘Anil’;
```

Instructor Notes:



4.4 DML command (Insert, Update, Delete)

UPDATE

- Use the UPDATE command to change single rows, groups of rows, or all rows in a table.
  - In all data modification statements, you can change the data in only “one table at a time”.

```
UPDATE table_name
SET   col_name = value|
       col_name =
SELECT_statement_returning_single_value|
       (col_name,...) = SELECT_statement
[WHERE condition];
```

Modifying / Updating existing Data in a Table:

- The table_name can be a “table” or a “view”.
- The “value” can be a value, an expression, or a query, which returns a single value.
- The UPDATE command provides automatic navigation to the data.
- **Note:** If the WHERE clause is omitted, all rows in the table will be updated by a value that is currently specified for the field. Else only those rows which satisfy the condition will be updated.

Instructor Notes:

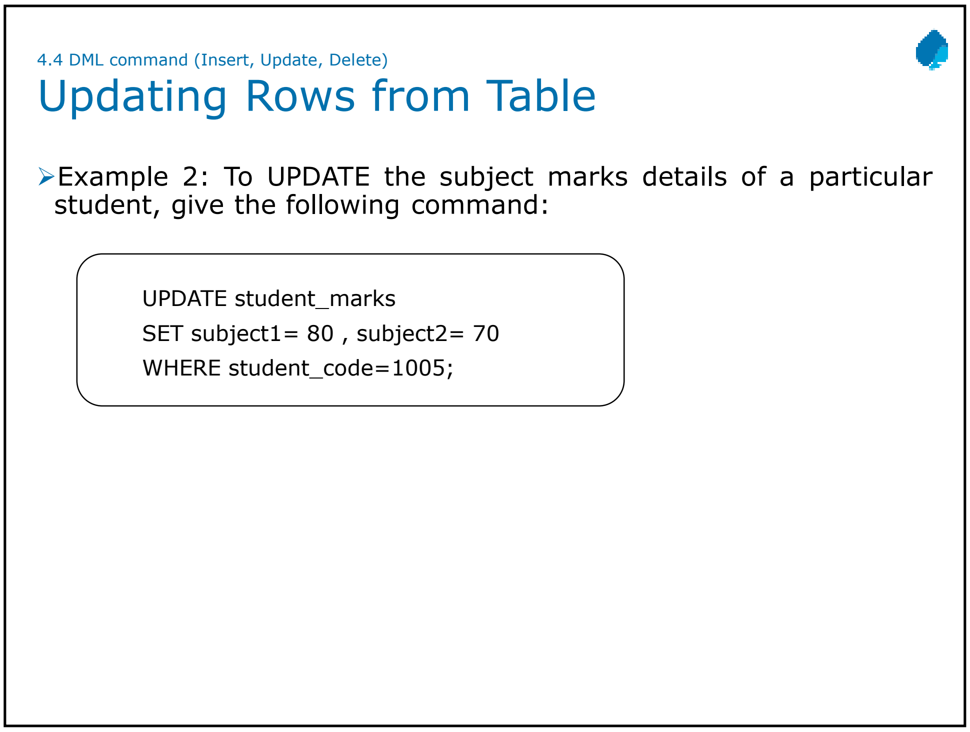
4.4 DML command (Insert, Update, Delete)

Updating Rows from Table

- Example 1: To UPDATE the column “dname” of a row, where deptno is 10, give the following command:

```
UPDATE department_master
SET dept_name= ‘Information Technology’
WHERE dept_code=10;
```

Instructor Notes:

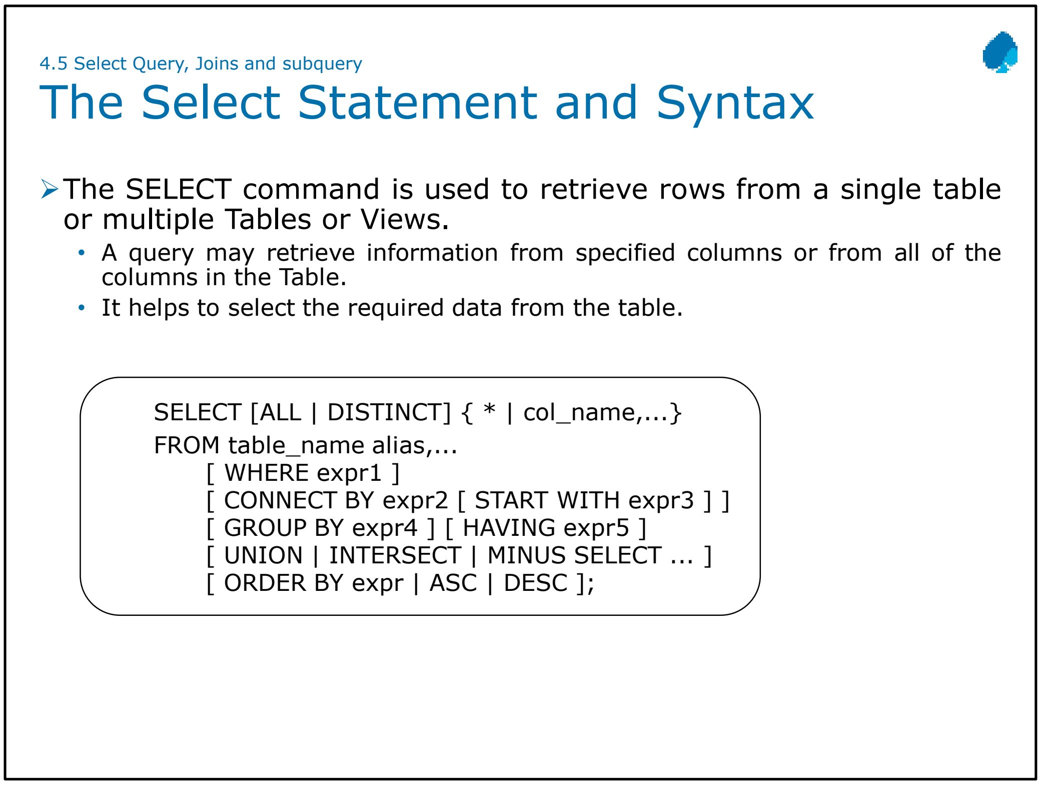


Updating Rows from Table

- Example 2: To UPDATE the subject marks details of a particular student, give the following command:

```
UPDATE student_marks
SET subject1= 80 , subject2= 70
WHERE student_code=1005;
```

Instructor Notes:



4.5 Select Query, Joins and subquery

The Select Statement and Syntax

- The SELECT command is used to retrieve rows from a single table or multiple Tables or Views.
  - A query may retrieve information from specified columns or from all of the columns in the Table.
  - It helps to select the required data from the table.

```
SELECT [ALL | DISTINCT] { * | col_name,...}
FROM table_name alias,...
     [ WHERE expr1 ]
     [ CONNECT BY expr2 [ START WITH expr3 ] ]
     [ GROUP BY expr4 ] [ HAVING expr5 ]
     [ UNION | INTERSECT | MINUS SELECT ... ]
     [ ORDER BY expr | ASC | DESC ];
```

The SELECT Statement:

- The SELECT statement is used to select data from a table. The tabular result is stored in a result table (called the result-set). The statement begins with the SELECT keyword. The basic SELECT statement has three clauses:
  - SELECT
  - FROM
  - WHERE
- The SELECT clause specifies the table columns that are retrieved.
- The FROM clause specifies the tables accessed.
- The WHERE clause specifies which table rows are used. The WHERE clause is optional; if missing, all table rows are used.

Note:

- Each clause is evaluated on the result set of a previous clause. The final result of the query will be always a “result table”.
- Only FROM clause is essential. The clauses WHERE, GROUP BY, HAVING, ORDER BY are optional.
- All the examples that follow are based on EMP and DEPT tables that are already available.

Instructor Notes:

Selecting Columns

- Displays all the columns from the student_master table

```
SELECT  *
        FROM student_master;
```

- Displays selected columns from the student_master table

```
SELECT student_code, student_name
        FROM student_master;
```

Instructor Notes:

The WHERE clause

- The WHERE clause is used to specify the criteria for selection.
  - For example: displays the selected columns from the student_master table based on the condition being satisfied

```
SELECT  student_code,  student_name, student_dob
                  FROM student_master
           WHERE dept_code = 10;
```

The WHERE Clause:

- The WHERE clause is used to perform "selective retrieval" of rows. It follows the FROM clause, and specifies the search condition.
- The result of the WHERE clause is the row or rows retrieved from the Tables, which meet the search condition.
- The clause is of the form:

```
WHERE <search condition>
```

Comparison Predicates:

- The Comparison Predicates specify the comparison of two values.
  - It is of the form:

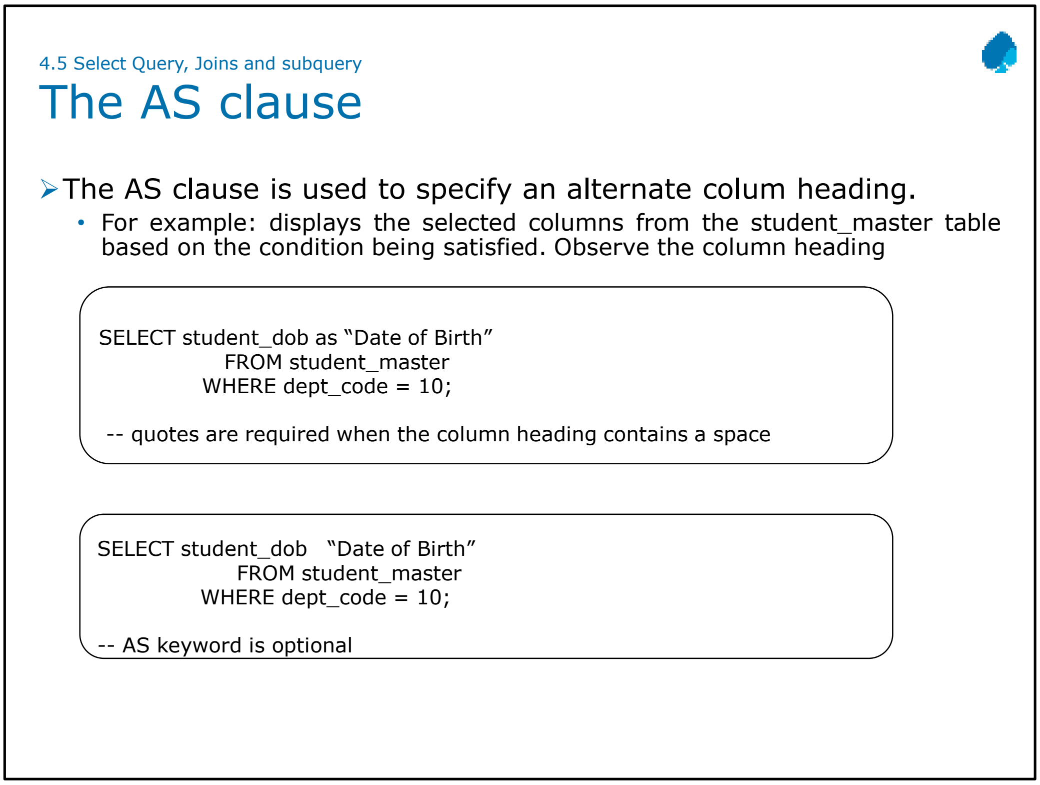
    < Expression> < operator > < Expression>

    < Expression> <operator> <subquery>
  - The operators used are shown on the next slide:

Instructo

contd.

Instructor Notes:



4.5 Select Query, Joins and subquery

The AS clause

- The AS clause is used to specify an alternate colum heading.
  - For example: displays the selected columns from the student_master table based on the condition being satisfied. Observe the column heading

```
SELECT student_dob as "Date of Birth"
                  FROM student_master
               WHERE dept_code = 10;

 -- quotes are required when the column heading contains a space
```

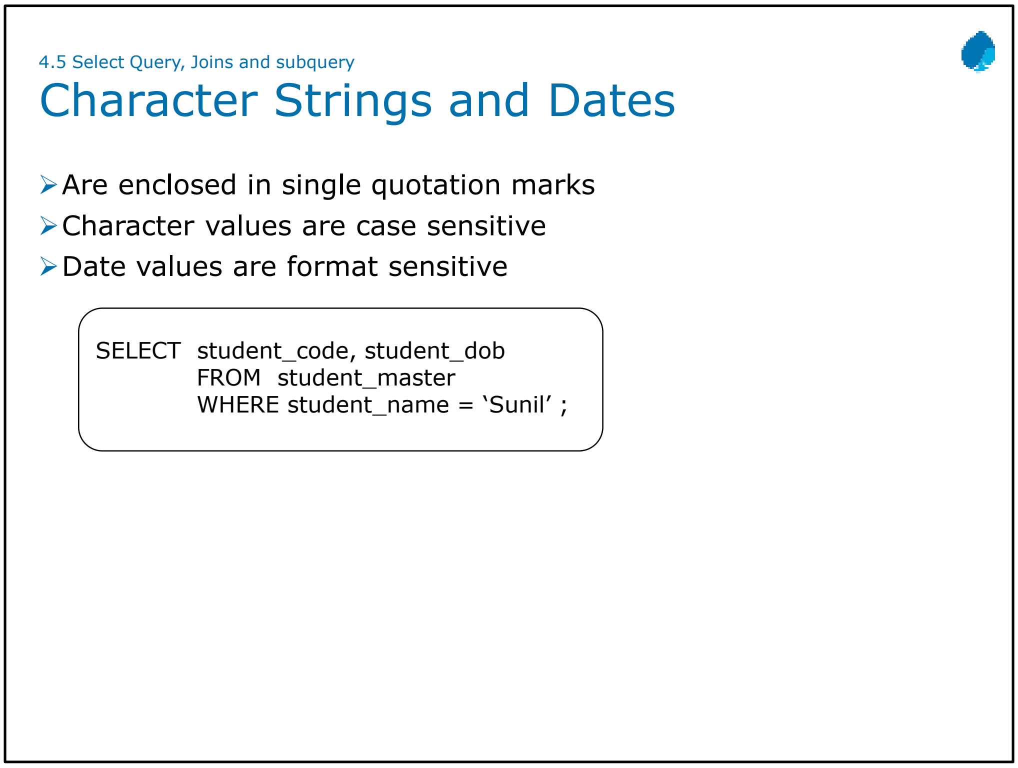
```
SELECT student_dob   "Date of Birth"
                    FROM student_master
               WHERE dept_code = 10;

-- AS keyword is optional
```

<u>The AS Clause</u>:

- The AS clause is used to give a different column heading (other than column name) to one or more columns used in the select statement. It follows the column name, and can be used for one or more columns.
- The AS keyword is optional.
- The clause is of the form:

```
Select column1  heading1, column2   as  heading1,
column3 as "heading3 contains space" from
table_name
```

Instructor Notes:



4.5 Select Query, Joins and subquery

Character Strings and Dates

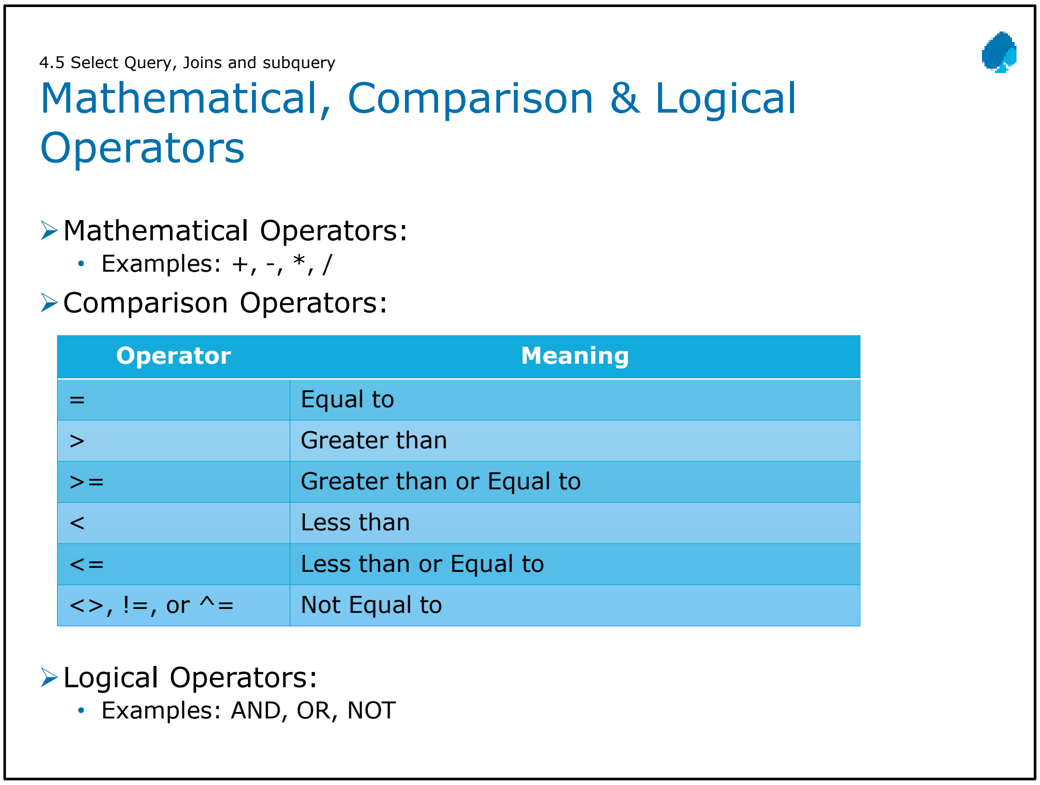
- Are enclosed in single quotation marks
- Character values are case sensitive
- Date values are format sensitive

```
SELECT  student_code, student_dob
        FROM  student_master
        WHERE student_name = ‘Sunil’ ;
```

Oracle Database store dates in an internal numeric format, representing the century, year, month, day, hours, minutes, and seconds.

The date datatype is covered in detail later.

Instructor Notes:



4.5 Select Query, Joins and subquery

Mathematical, Comparison & Logical Operators

- Mathematical Operators:
  - Examples: +, -, *, /
- Comparison Operators:

Operator	Meaning
=	Equal to
>	Greater than
>=	Greater than or Equal to
<	Less than
<=	Less than or Equal to
<>, !=, or ^=	Not Equal to

- Logical Operators:
  - Examples: AND, OR, NOT

<u>Operators</u>:

- Operators are used in “expressions” or “conditional statements”. They show equality, inequality, or a combination of both.
- Operators are of three types:
  - mathematical
  - logical
  - range (comparison)
- These operators are mainly used in the WHERE clause, HAVING clause in order to filter the data to be selected.
- **Mathematical operators:**

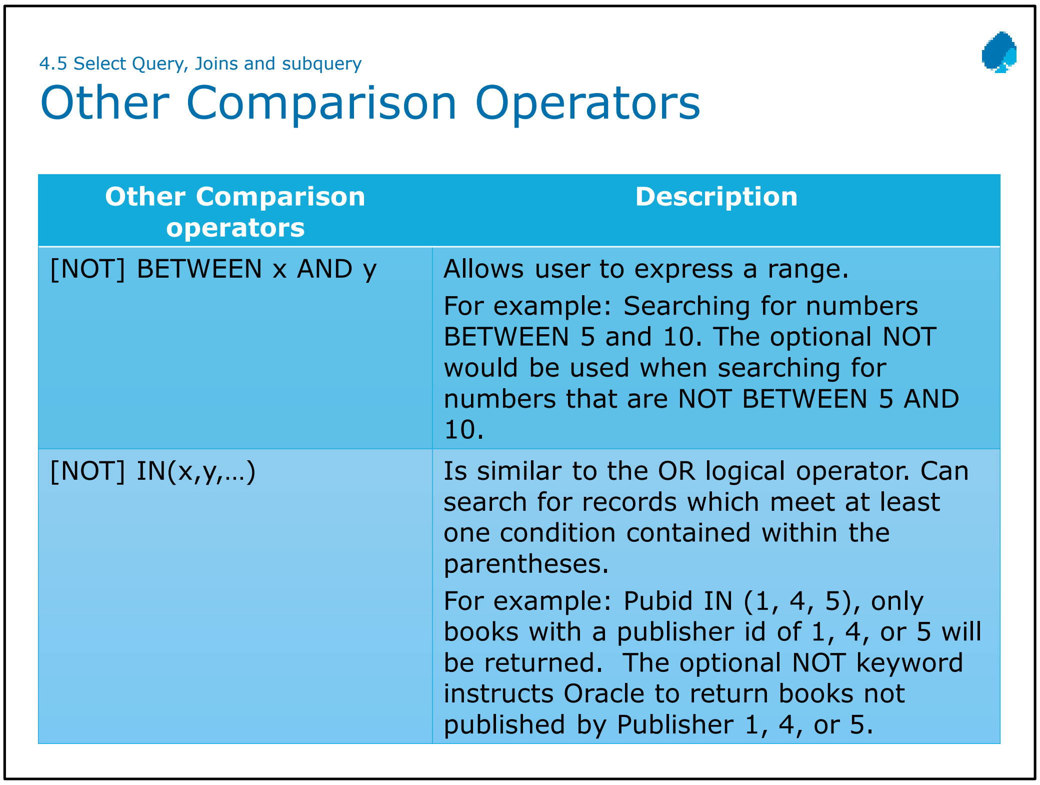
  These operators add, subtract, multiply, divide, and compare equality of numbers and strings. They are +, -, *, /
- **Comparison Operators:**

  These operators are used to compare the column data with specific values in a condition. “Comparison Operators” are also used along with the “SELECT statement” to filter data based on specific conditions. The table in the slide describes each Comparison operator. Comparison operators indicate how the data should relate to the given search value.
- **Logical Operators:**

  There are three Logical Operators namely AND, OR and NOT. These operators compare two conditions at a time to determine whether a row can

Instructo be selected for the output or not. When retrieving data by using a SELECT statement, you can use logical operators in the WHERE clause. This allows you to combine more than one condition.

Instructor Notes:



Other Comparison Operators

Other Comparison operators	Description
[NOT] BETWEEN x AND y	Allows user to express a range. For example: Searching for numbers BETWEEN 5 and 10. The optional NOT would be used when searching for numbers that are NOT BETWEEN 5 AND 10.
[NOT] IN(x,y,…)	Is similar to the OR logical operator. Can search for records which meet at least one condition contained within the parentheses. For example: Pubid IN (1, 4, 5), only books with a publisher id of 1, 4, or 5 will be returned. The optional NOT keyword instructs Oracle to return books not published by Publisher 1, 4, or 5.

Instructor Notes:

Other Comparison Operators

Other Comparison operators	Description
[NOT] LIKE	Can be used when searching for patterns if you are not certain how something is spelt. For example: title LIKE 'TH%'. Using the optional NOT indicates that records that do contain the specified pattern should not be included in the results.
IS[NOT]NULL	Allows user to search for records which do not have an entry in the specified field. For example: Shipdate IS NULL. If you include the optional NOT, it would find the records that do not have an entry in the field. For example: Shipdate IS NOT NULL.

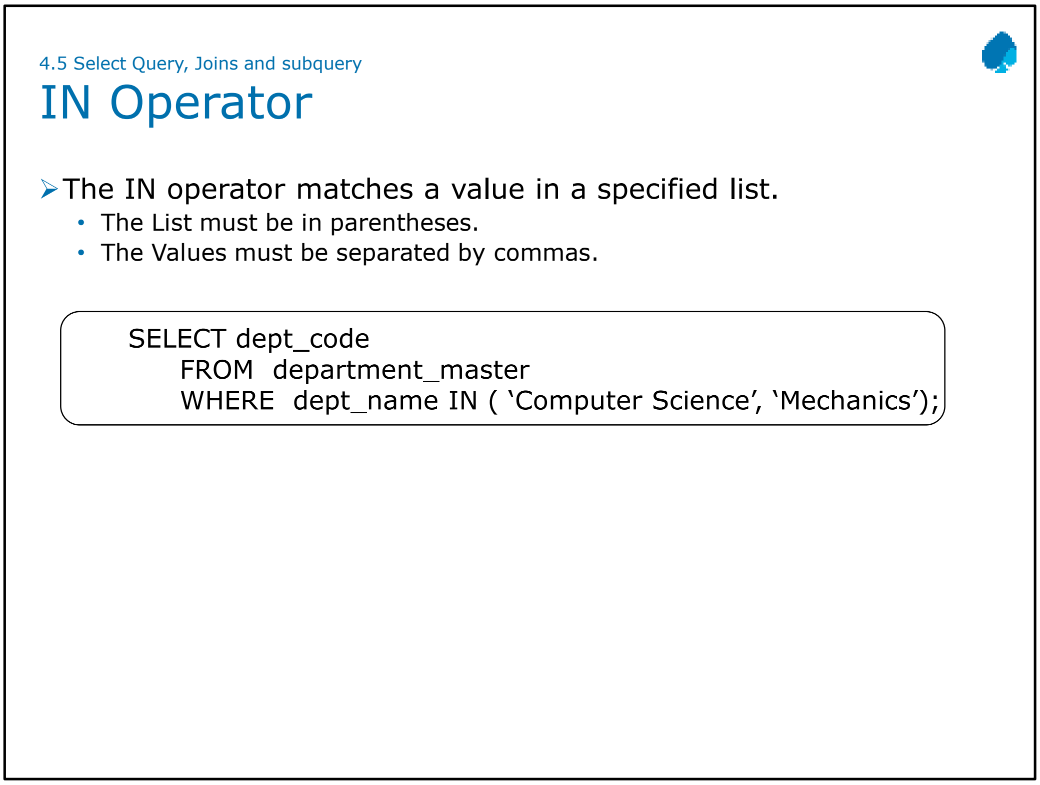
Instructor Notes:

BETWEEN … AND Operator

- The BETWEEN … AND operator finds values in a specified range:

```
SELECT staff_code,staff_name
     FROM  staff_master
     WHERE  staff_dob  BETWEEN ’01-Jan-1980’
            AND  ’31-Jan-1980’;
```

Instructor Notes:



4.5 Select Query, Joins and subquery

IN Operator

- The IN operator matches a value in a specified list.
  - The List must be in parentheses.
  - The Values must be separated by commas.

```
SELECT dept_code
     FROM  department_master
     WHERE  dept_name IN ( ‘Computer Science’, ‘Mechanics’);
```

IN predicate:

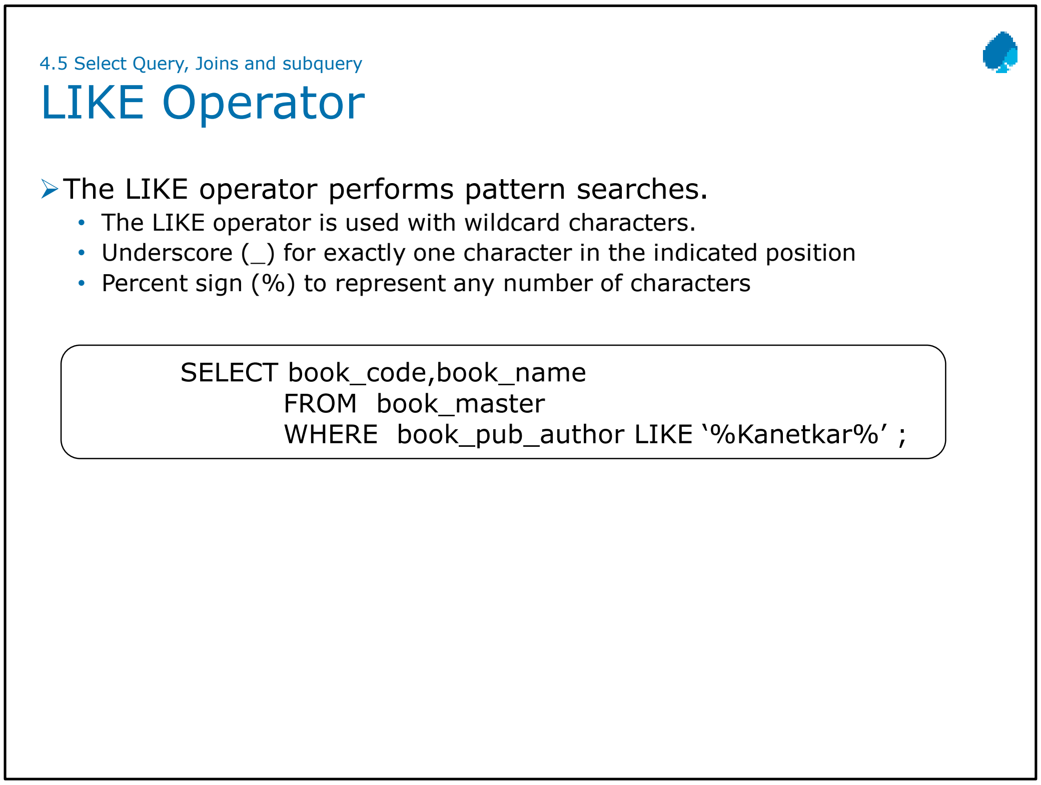
- It is of the form:

  <Expression> IN <LIST>

  <Expression> IN <SUBQUERY>

- The data types should match.

Instructor Notes:



4.5 Select Query, Joins and subquery

LIKE Operator

- The LIKE operator performs pattern searches.
  - The LIKE operator is used with wildcard characters.
  - Underscore (_) for exactly one character in the indicated position
  - Percent sign (%) to represent any number of characters

```
SELECT book_code,book_name
        FROM   book_master
        WHERE   book_pub_author LIKE '%Kanetkar%' ;
```

LIKE predicate:

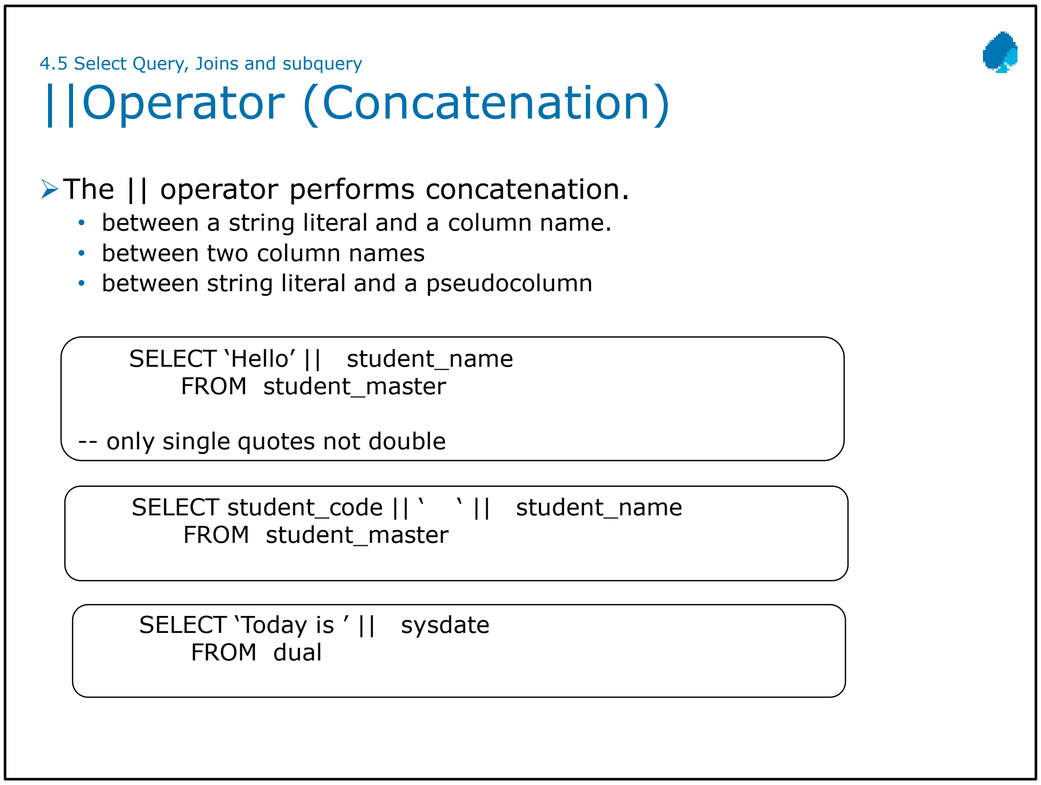
- It is of the form:

  <COLUMN > LIKE < PATTERN>

- The pattern contains a search string along with other special characters % and _. The % character represents a string of any length where as _ (underscore) represents exactly one character.
- A pattern %XYZ% means search has to be made for string XYZ in any position. A pattern '_XYZ%' means search has to be made for string XYZ in position 2 to 4.
- To search for characters % and _ in the string itself we have to use an "escape" character.

  For example: To search for string NOT_APP in column status, we have to use the form Status like 'NOT_APP' ESCAPE '\'
- The use of \ as escape character is purely arbitrary.

Instructor Notes:



4.5 Select Query, Joins and subquery

||Operator (Concatenation)

- The || operator performs concatenation.
  - between a string literal and a column name.
  - between two column names
  - between string literal and a pseudocolumn

```
     SELECT ‘Hello’ ||   student_name
          FROM  student_master

-- only single quotes not double
```

```
     SELECT student_code || ‘    ‘ ||   student_name
          FROM  student_master
```

```
     SELECT ‘Today is ’ ||   sysdate
          FROM  dual
```

Retrieval of Constant values by using Dual Table

A “dual” is a table, which is created by Oracle along with the data dictionary. It consists of exactly one column, whose name is dummy, and one record. The value of that record is X.

```
SQL>desc dual;
Name                          Null?    Type
DUMMY                                  VARCHAR2(1)
Sql>Select * from dual;
D
-
X
```

The owner of dual is SYS. However, “dual” can be accessed by every user.

As “dual” contains exactly one row (unless someone has fiddled with it), it is guaranteed to return exactly one row in SELECT statements.

```
SQL>select sysdate from dual;
```

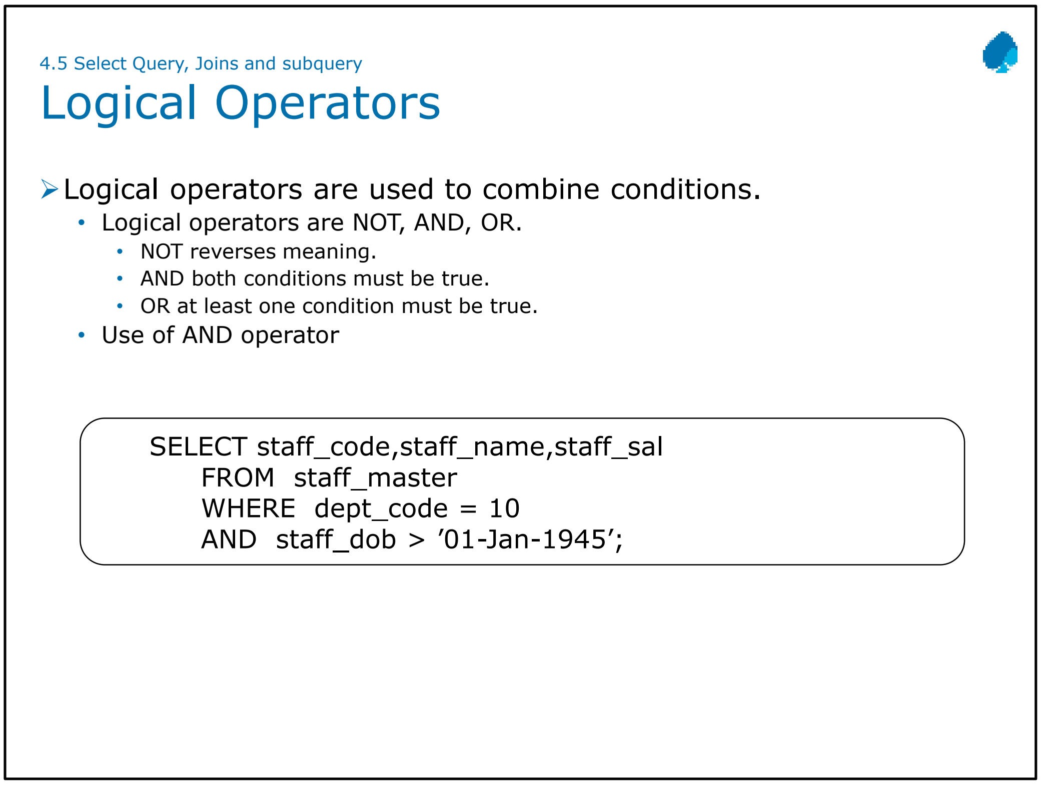
For example, you can use it for math:

```
SQL>SELECT (319/212)+10 FROM DUAL;
```

And, you can use it to increment sequences:

```
SQL>SELECT employee_seq.NEXTVAL FROM
DUAL;
```

Instructor Notes:



Logical Operators

- Logical operators are used to combine conditions.
  - Logical operators are NOT, AND, OR.
    - NOT reverses meaning.
    - AND both conditions must be true.
    - OR at least one condition must be true.
  - Use of AND operator

```
SELECT staff_code,staff_name,staff_sal
     FROM  staff_master
     WHERE  dept_code = 10
     AND  staff_dob > '01-Jan-1945';
```

The AND operator displays a record if both the first condition and the second condition is true.

One More Example:

```
SQL>  SELECT title, pubid, category
  2   FROM books
  3   WHERE pubid = 3
  4   AND category = 'COMPUTER';
```

Combining Predicates by using Logical Operators:

- The predicates can be combined by using logical operators like AND, OR, NOT. The evaluation proceeds from left to right and order of evaluation is:
  - * Enclosed in parenthesis
  - AND
  - OR

Instructor Notes:

Using AND or OR Clause

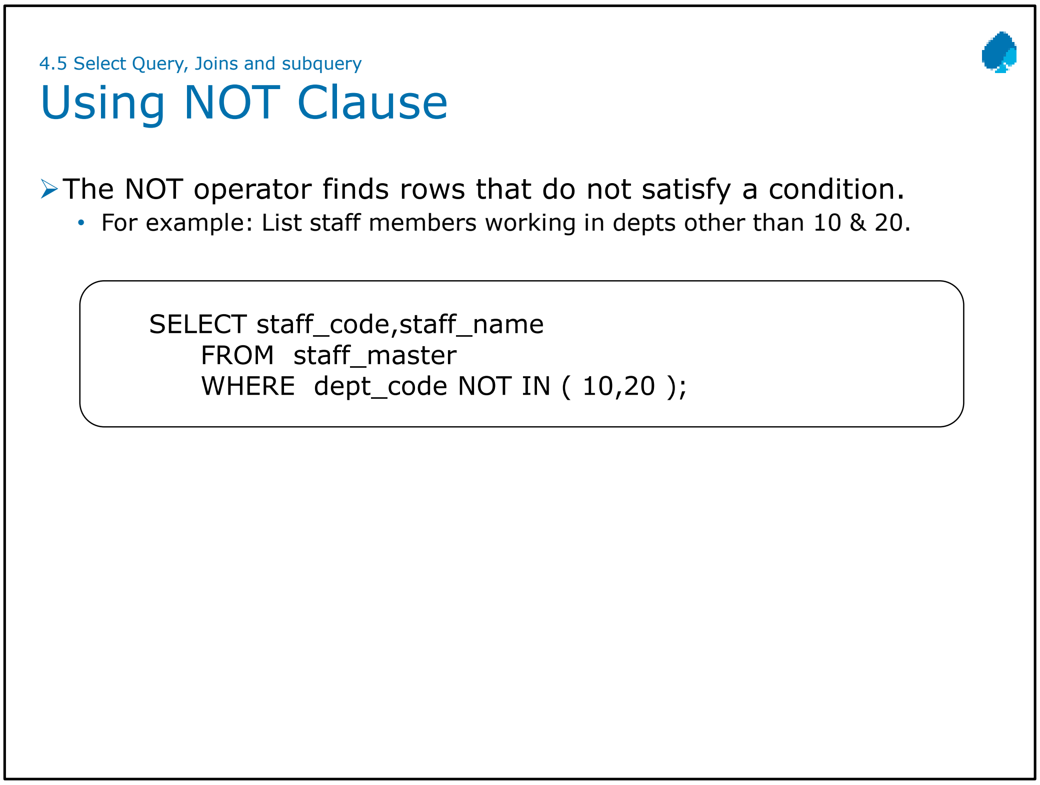
- Use of OR operator:

```
SELECT  book_code
    FROM  book_master
    WHERE  book_pub_author LIKE '%Kanetkar%'
    OR  book_name LIKE '%Pointers%';
```

The OR operator displays a record if either the first condition or the second condition is true.

You can also combine AND and OR as shown in above example. (use parenthesis to form complex expressions).

Instructor Notes:



4.5 Select Query, Joins and subquery

Using NOT Clause

- The NOT operator finds rows that do not satisfy a condition.
  - For example: List staff members working in depts other than 10 & 20.

```
SELECT staff_code,staff_name
    FROM  staff_master
    WHERE  dept_code NOT IN ( 10,20 );
```

- **Note:** NOT is a negation operator.

Instructor Notes:

Treatment of NULL Values

- NULL is the absence of data.
- Treatment of this scenario requires use of IS NULL operator.

```
SQL>SELECT student_code
          FROM student_master
          WHERE dept_code IS NULL;
```

NULL predicate:

The NULL predicate specifies a test for NULL values. The form for NULL predicate is:

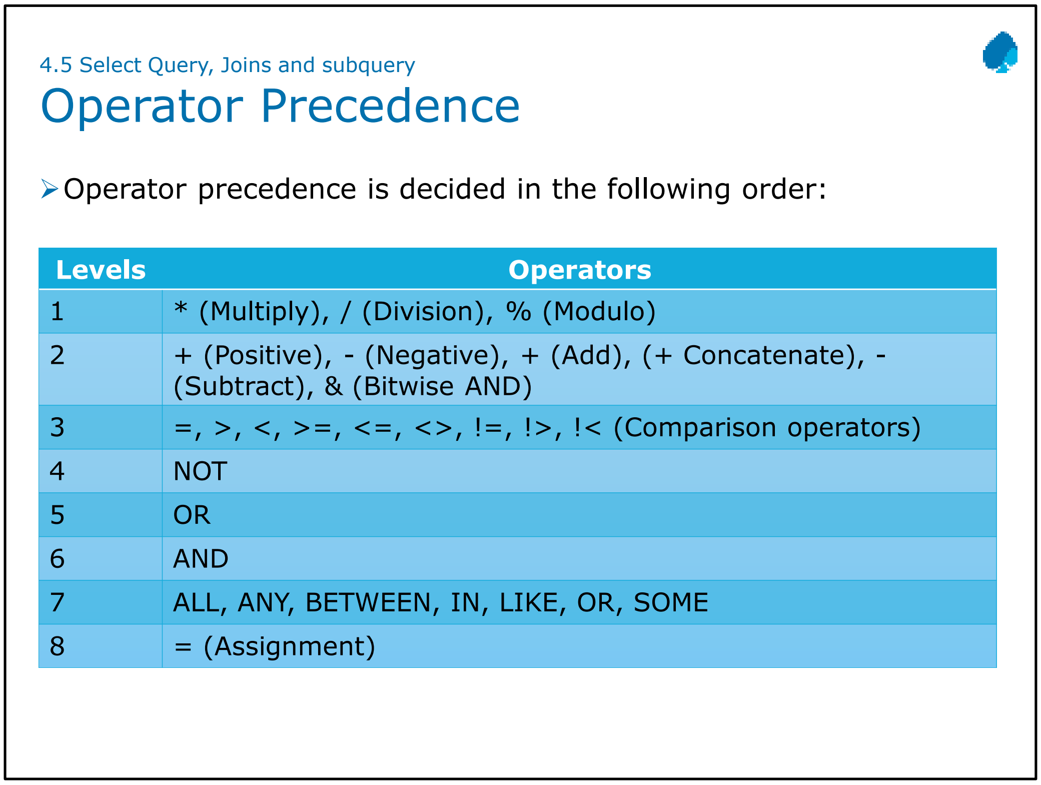
< COLUMN SPECIFICATION > IS NULL.

< COLUMN SPECIFICATION > IS NOT NULL.

< COLUMN SPECIFICATION > IS NULL returns TRUE only when column has NULL values.

<COLUMN> = NULL cannot be used to compare null values.

Instructor Notes:



4.5 Select Query, Joins and subquery

Operator Precedence

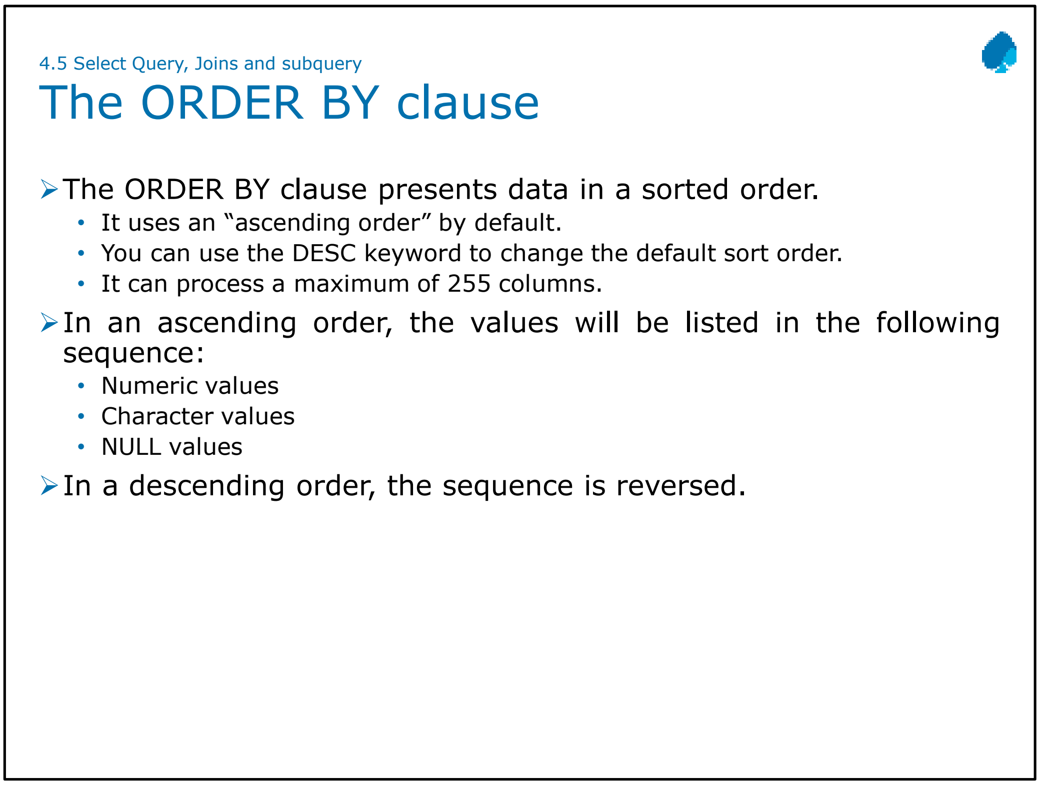
- Operator precedence is decided in the following order:

Levels	Operators
1	* (Multiply), / (Division), % (Modulo)
2	+ (Positive), - (Negative), + (Add), (+ Concatenate), - (Subtract), & (Bitwise AND)
3	=, >, <, >=, <=, <>, !=, !>, !< (Comparison operators)
4	NOT
5	OR
6	AND
7	ALL, ANY, BETWEEN, IN, LIKE, OR, SOME
8	= (Assignment)

Operator Precedence:

- When a complex expression has multiple operators, the operator precedence (or order of execution of operators) determines the sequence in which the operations are performed.
- The order of execution can significantly affect the resulting value.
- The operators have the precedence levels as shown in the table given in the slide.
- An operator on higher levels is evaluated before an operator on lower level.

Instructor Notes:



The ORDER BY clause

- The ORDER BY clause presents data in a sorted order.
  - It uses an "ascending order" by default.
  - You can use the DESC keyword to change the default sort order.
  - It can process a maximum of 255 columns.
- In an ascending order, the values will be listed in the following sequence:
  - Numeric values
  - Character values
  - NULL values
- In a descending order, the sequence is reversed.

<u>The Order By Clause</u>:

- A query with its various clauses (FROM, WHERE, GROUP BY, HAVING) determines the rows to be selected and the columns. The order of rows is not fixed unless an ORDER BY clause is given.
- An ORDER BY clause is of the form:

```
ORDER BY < Sort  list> ASC/DESC
```

- The columns to be used for ordering are specified by using the "column names" or by specifying the "serial number" of the column in the SELECT list.
- The sort is done on the column in "ascending" or "descending" order. By default the ordering of data is "ascending" order.

contd.

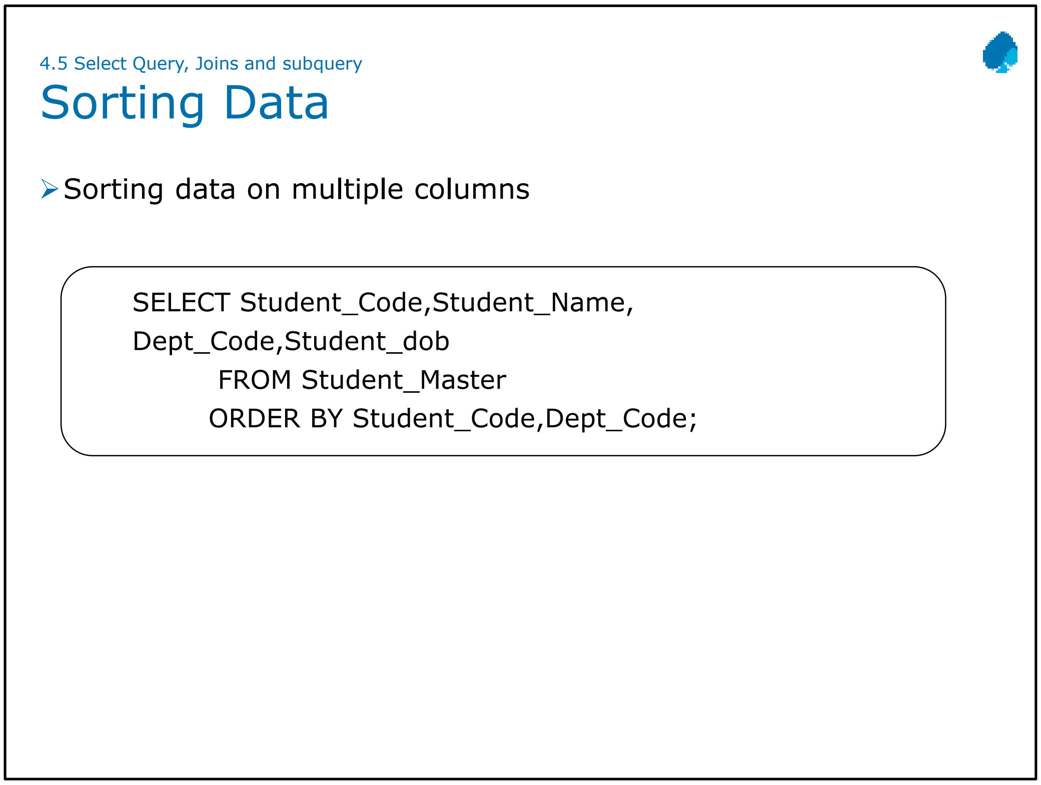
Instructor Notes:

Sorting Data

- The output of the SELECT statement can be sorted using ORDER BY clause
  - ASC : Ascending order, default
  - DESC : Descending order
- Display student details from student_master table sorted on student_code in descending order.

```
SELECT Student_Code,Student_Name,Dept_Code,
Student_dob
      FROM Student_Master
     ORDER BY Student_Code DESC ;
```

Instructor Notes:



4.5 Select Query, Joins and subquery

Sorting Data

- Sorting data on multiple columns

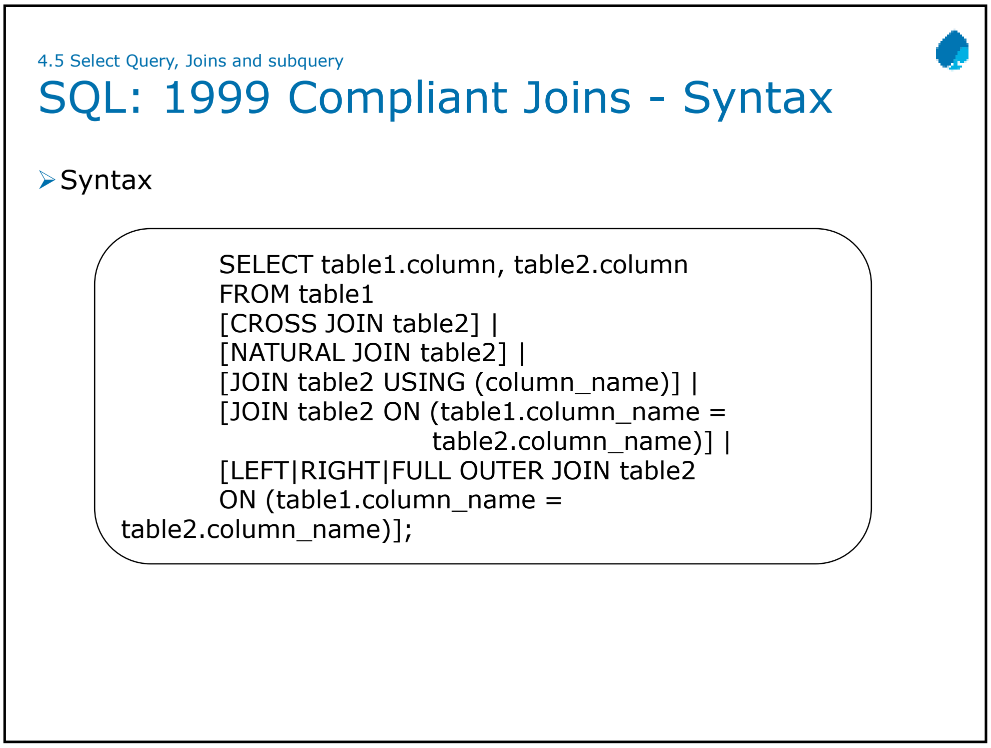
```
SELECT Student_Code,Student_Name,
Dept_Code,Student_dob
         FROM Student_Master
        ORDER BY Student_Code,Dept_Code;
```

The query on the slide sorts the data on both the columns in ascending order which is default. But you could also sort the data in different order for the columns.

For Example in the query given below the data is sorted in ascending order on student_code and dept_code is sorted in descending order

```
SELECT
Student_Code,Student_Name,Dept_Code,Student_dob
  FROM Student_Master
 ORDER BY Student_Code,Dept_Code DESC;
```

Instructor Notes:



4.5 Select Query, Joins and subquery

SQL: 1999 Compliant Joins - Syntax

- Syntax

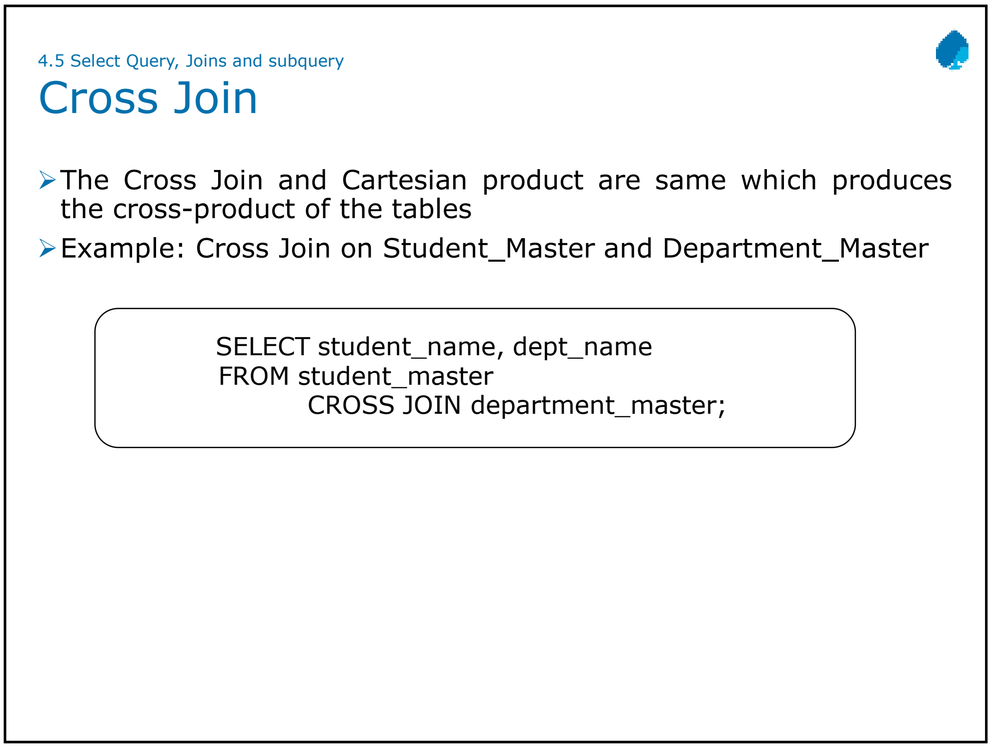
```
SELECT table1.column, table2.column
FROM table1
[CROSS JOIN table2] |
[NATURAL JOIN table2] |
[JOIN table2 USING (column_name)] |
[JOIN table2 ON (table1.column_name =
                 table2.column_name)] |
[LEFT|RIGHT|FULL OUTER JOIN table2
ON (table1.column_name =
table2.column_name)];
```

<u>SQL:1999 Compliant Joins</u>

The SQL Compliant joins can obtain the similar results that we have discussed in the previous slides. They differ only in syntax.

The SQL compliant joins are supported from Oracle 9i version onwards

Instructor Notes:



4.5 Select Query, Joins and subquery

Cross Join

- The Cross Join and Cartesian product are same which produces the cross-product of the tables
- Example: Cross Join on Student_Master and Department_Master

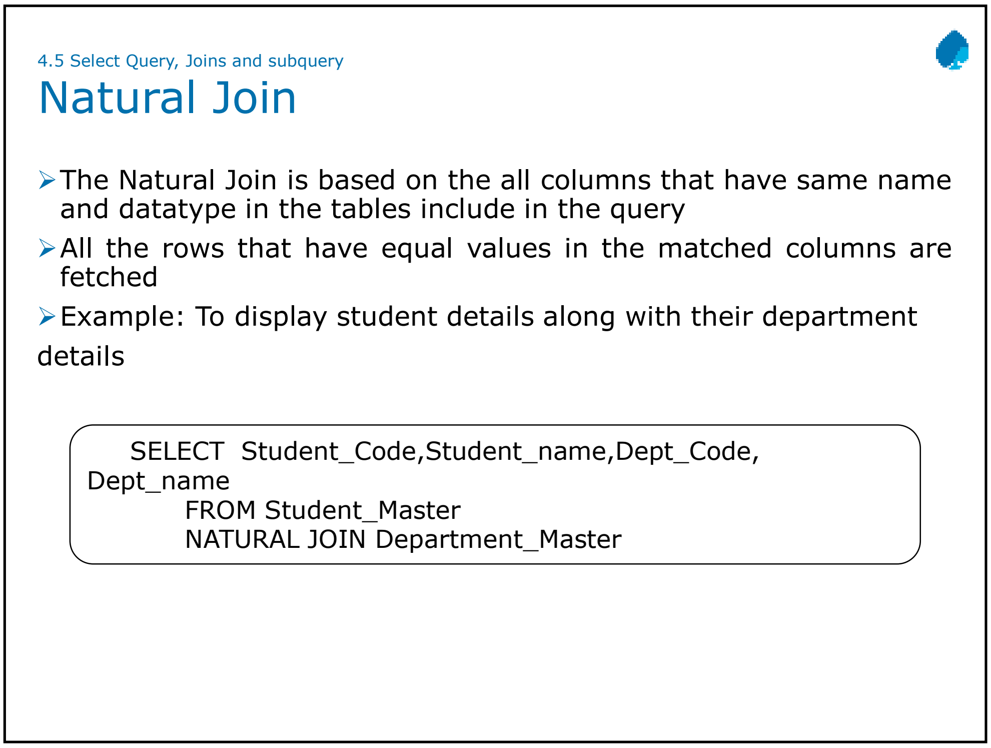
```
SELECT student_name, dept_name
 FROM student_master
          CROSS JOIN department_master;
```

Cross Join

• The example on the slide create a cross product of the two tables. The query result is same as the following query:

```
SELECT student_name,dept_name
FROM Student_Master, Department_Master;
```

Instructor Notes:



4.5 Select Query, Joins and subquery

Natural Join

- The Natural Join is based on the all columns that have same name and datatype in the tables include in the query
- All the rows that have equal values in the matched columns are fetched
- Example: To display student details along with their department details

```
     SELECT  Student_Code,Student_name,Dept_Code,
Dept_name
          FROM Student_Master
          NATURAL JOIN Department_Master
```

Natural Join

Prior to Oracle 9i, without explicitly specifying the columns of the corresponding tables a join was not possible. With the Natural Join clause, the join can happen automatically based on column names that match in name and datatype.

Oracle returns an error if the datatypes of the columns are different though they have the same name.

The Natural Join is same as EquiJoin.

Instructor Notes:

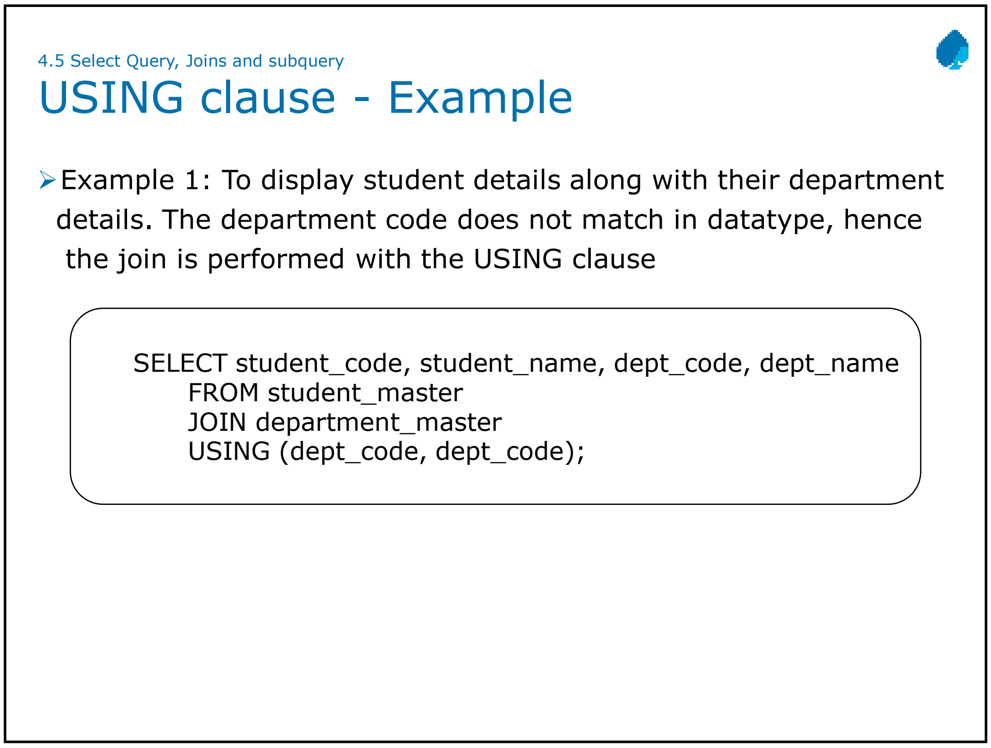
USING clause

- The USING clause can be replace the NATURAL JOIN if the columns have same names but data types do not match.
- The table name or aliases should not be used in the referenced columns
- This clause should be used to match only one column when there are more than one column matches

USING clause

- The NATURAL JOIN returns an error if the datatypes of matching columns are different. In that case the USING clause can be used to specify only those columns on which the join has to done.
- The NATURAL JOIN and USING clause are mutually exclusive.
- The column used in USING clause cannot be used in WHERE clause

Instructor Notes:



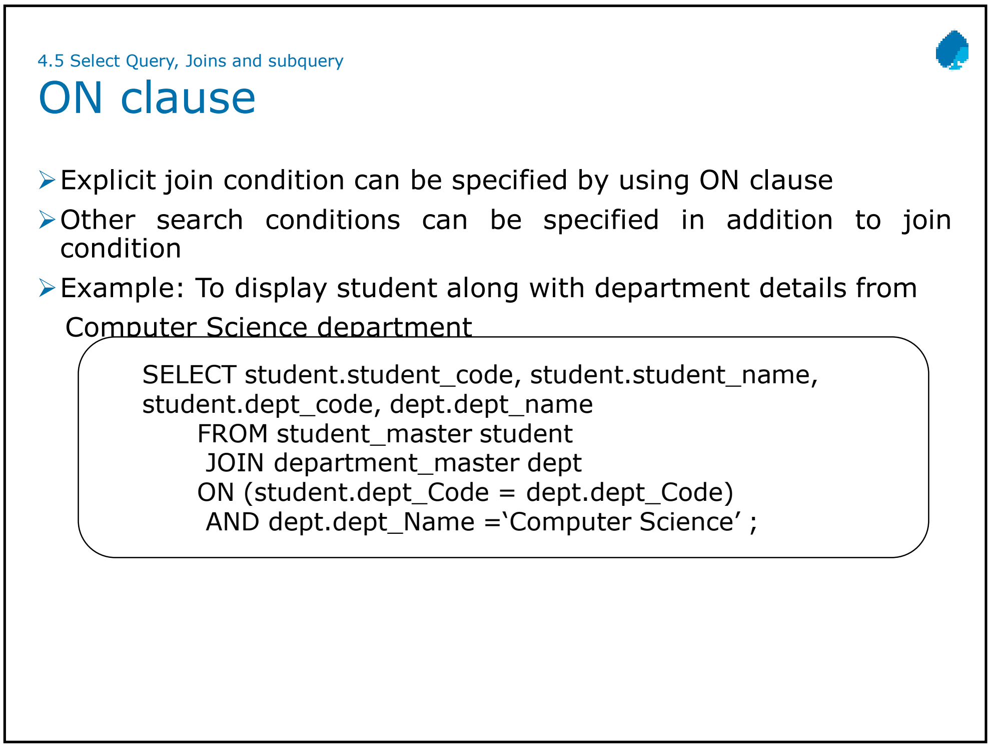
4.5 Select Query, Joins and subquery

USING clause - Example

- Example 1: To display student details along with their department details. The department code does not match in datatype, hence the join is performed with the USING clause

```
SELECT student_code, student_name, dept_code, dept_name
      FROM student_master
      JOIN department_master
      USING (dept_code, dept_code);
```

Instructor Notes:



ON clause

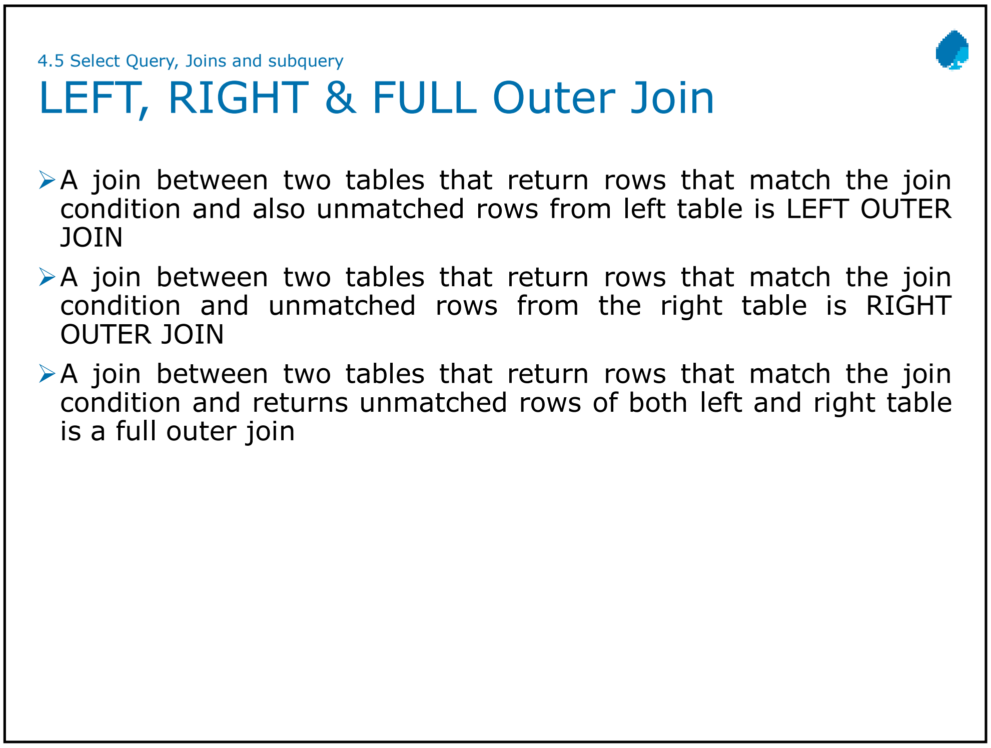
- Explicit join condition can be specified by using ON clause
- Other search conditions can be specified in addition to join condition
- Example: To display student along with department details from Computer Science department

```
SELECT student.student_code, student.student_name,
student.dept_code, dept.dept_name
        FROM student_master student
         JOIN department_master dept
        ON (student.dept_Code = dept.dept_Code)
         AND dept.dept_Name ='Computer Science' ;
```

ON clause

- With the ON clause you can specify join conditions separate from any other search conditions.
- The query is readable and easy to understand

Instructor Notes:



4.5 Select Query, Joins and subquery

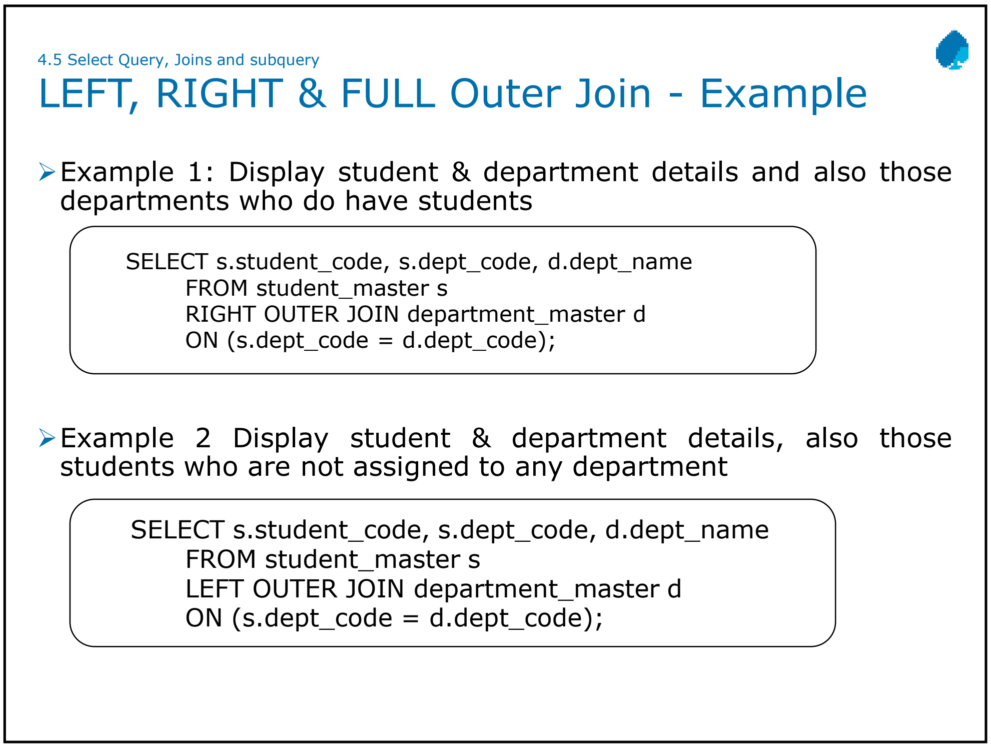
LEFT, RIGHT & FULL Outer Join

- A join between two tables that return rows that match the join condition and also unmatched rows from left table is LEFT OUTER JOIN
- A join between two tables that return rows that match the join condition and unmatched rows from the right table is RIGHT OUTER JOIN
- A join between two tables that return rows that match the join condition and returns unmatched rows of both left and right table is a full outer join

LEFT, RIGHT and FULL OUTER JOIN

The SQL compliant outer join is similar to Oracle proprietary outer join except for the fact that it also has support to perform FULL OUTER JOIN

Instructor Notes:



LEFT, RIGHT & FULL Outer Join - Example

- Example 1: Display student & department details and also those departments who do have students

```
SELECT s.student_code, s.dept_code, d.dept_name
        FROM student_master s
        RIGHT OUTER JOIN department_master d
        ON (s.dept_code = d.dept_code);
```

- Example 2 Display student & department details, also those students who are not assigned to any department

```
SELECT s.student_code, s.dept_code, d.dept_name
        FROM student_master s
        LEFT OUTER JOIN department_master d
        ON (s.dept_code = d.dept_code);
```

Instructor Notes:

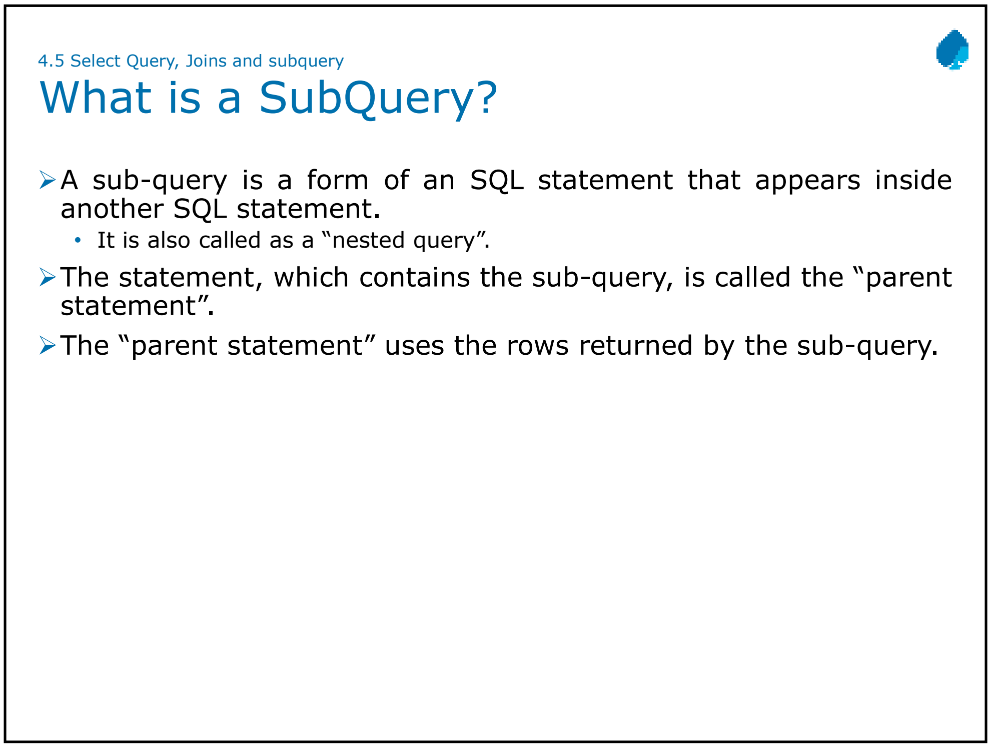
LEFT, RIGHT & FULL Outer Join - Example

➢Example 3: Display student & department details. Also those departments who do have students and students who are not assigned to any department

```
SELECT s.student_code,s.dept_code,d.dept_name
       FROM student_master s
       FULL OUTER JOIN department_master d
       ON (s.dept_code = d.dept_code );
```

Instructor Notes:

None



4.5 Select Query, Joins and subquery

What is a SubQuery?

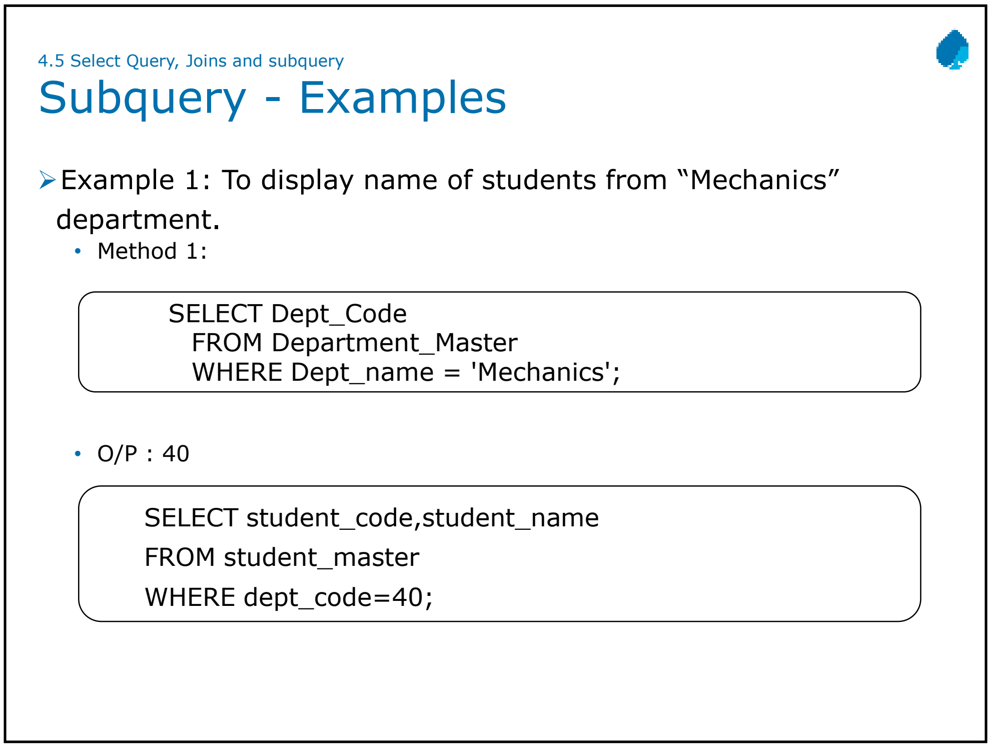
- A sub-query is a form of an SQL statement that appears inside another SQL statement.
  - It is also called as a "nested query".
- The statement, which contains the sub-query, is called the "parent statement".
- The "parent statement" uses the rows returned by the sub-query.

Sub-queries:

- As mentioned earlier, since a basic SQL query returns a relation, it can be used to construct composite queries.
- Such a SQL query, which is nested within another higher level query, is called a "sub-query".
- This kind of a nested sub-query is useful in cases, where we need to select rows from tables based on a condition, which depends on the data stored in the table itself.
- These can be useful when you need to select rows from a table with a condition that depends on data within the table itself.

Instructor Notes:

None



4.5 Select Query, Joins and subquery

Subquery - Examples

- Example 1: To display name of students from "Mechanics" department.
  - Method 1:

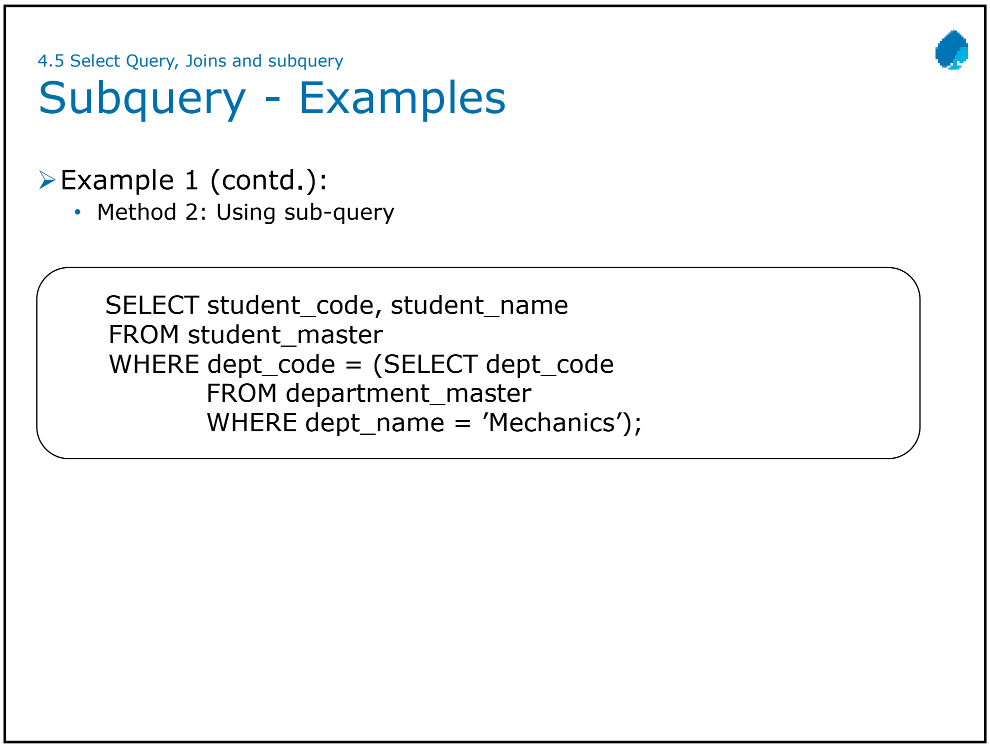
```
SELECT Dept_Code
  FROM Department_Master
  WHERE Dept_name = 'Mechanics';
```

  - O/P : 40

```
SELECT student_code,student_name
FROM student_master
WHERE dept_code=40;
```

Consider the example given on the slide. We want to find details of students from "Mechanics" department. As you can see two queries are used to resolve these problem. Instead of that you can resolve this problem by combining both the queries into one as shown on the next slide.

Instructor Notes:



4.5 Select Query, Joins and subquery

Subquery - Examples

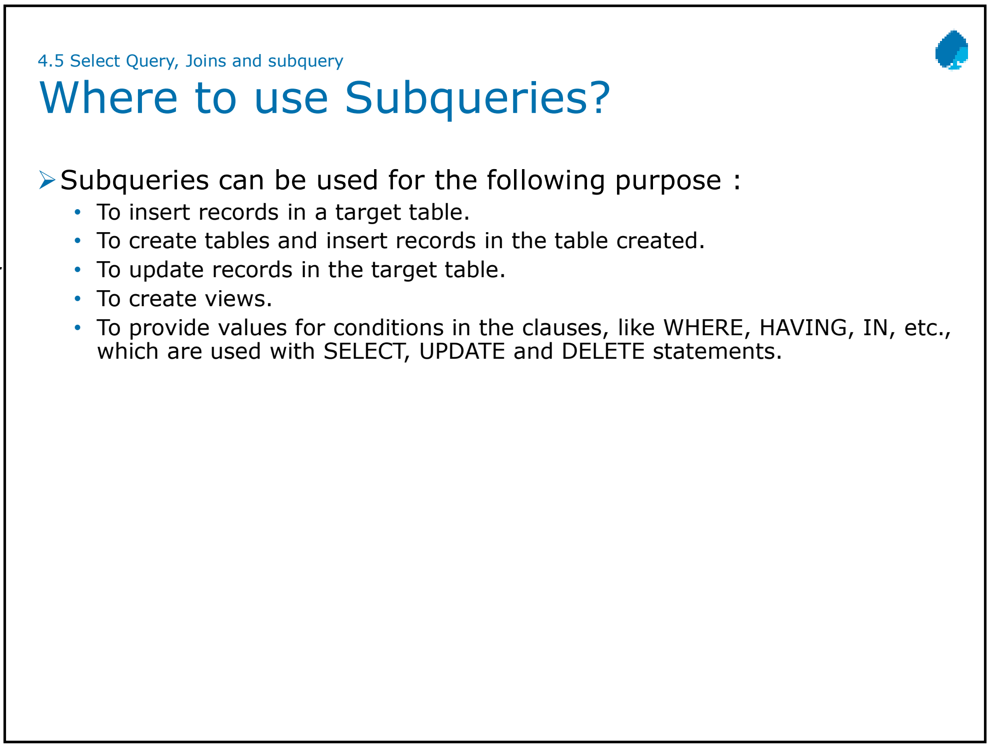
- Example 1 (contd.):
  - Method 2: Using sub-query

```
SELECT student_code, student_name
 FROM student_master
 WHERE dept_code = (SELECT dept_code
            FROM department_master
            WHERE dept_name = 'Mechanics');
```

The problem is resolved using a one query inside another. The inner query is called as a subquery or nested query. The subquery result is used by the outer query. The subqueries can have more levels of nesting. The innermost query will execute first. The subquery execute only once before the outer query

Instructor Notes:

Tell the participants that in this lesson we will only see subqueries in Select statement clauses. Others we will cover later like creating views, creating tables etc...



4.5 Select Query, Joins and subquery

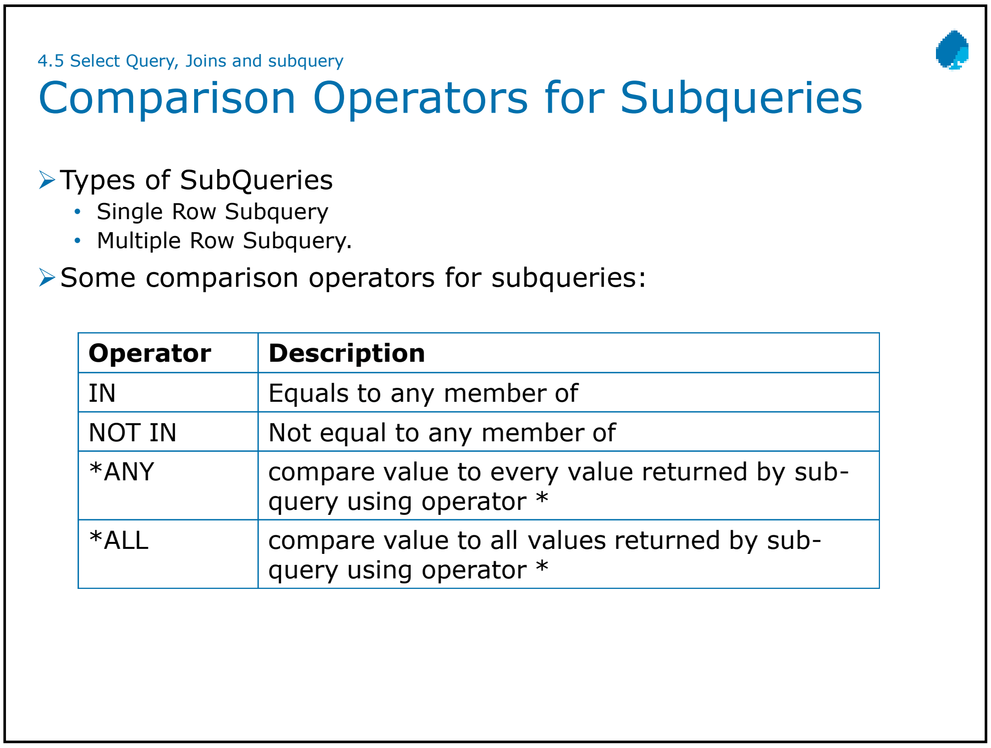
Where to use Subqueries?

- Subqueries can be used for the following purpose :
  - To insert records in a target table.
  - To create tables and insert records in the table created.
  - To update records in the target table.
  - To create views.
  - To provide values for conditions in the clauses, like WHERE, HAVING, IN, etc., which are used with SELECT, UPDATE and DELETE statements.

- When the WHERE clause needs a set of values which can be only obtained from another query, the Sub-query is used. In the WHERE clause it can become a part of the following predicates
  - COMPARISON Predicate
  - IN Predicate
  - ANY or ALL Predicate
  - EXISTS Predicate.
- It can be also used as a part of the condition in the HAVING clause.

Instructor Notes:

None



Comparison Operators for Subqueries

- Types of SubQueries
  - Single Row Subquery
  - Multiple Row Subquery.
- Some comparison operators for subqueries:

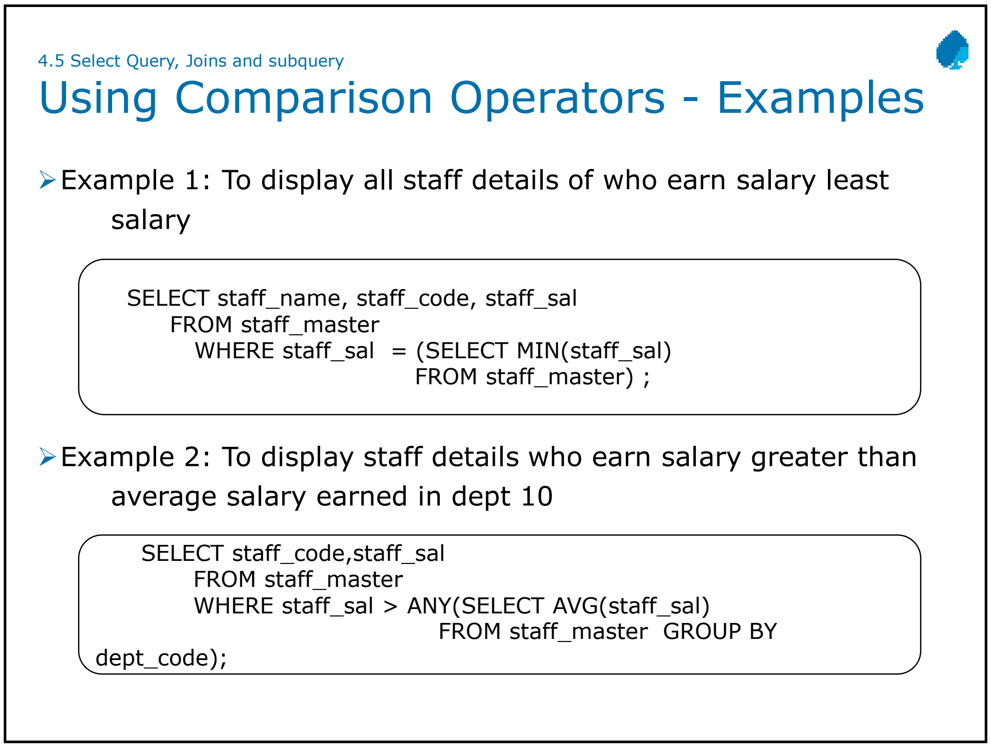
Operator	**Description**
IN	Equals to any member of
NOT IN	Not equal to any member of
*ANY	compare value to every value returned by sub-query using operator *
*ALL	compare value to all values returned by sub-query using operator *

<u>Sub-queries by using Comparison operators:</u>

- Sub-queries are divided as “single row” and “multiple row” sub-queries. While single row comparison operators (=, >, >=, <, <=, <>) can only be used in single row sub-queries, multiple row sub-queries can use IN, ANY or ALL.
- **For example**: The assignment operator (=) compares a single value to another single value. In case a value needs to be compared to a list of values, then the IN predicate is used. The IN predicate helps reduce the need to use multiple OR conditions.
- The NOT IN predicate is the opposite of the IN predicate. This will select all rows where values do not match the values in the list.
- The FOR ALL predicate is evaluated as follows:
  1. True if the comparison is true for every value of the list of values.
  2. True if sub-query gives a null set (No values)
  3. False if the comparison is false for one or more of the list of values generated by the sub-query.
- The FOR ANY predicate is evaluated as follows
  1. True if the comparison is true for one or more values generated by the sub-query.
  2. False if sub-query gives a null set (No values).
  3. False if the comparison is false for every value of the list of values generated by the sub-query.

Instructor Notes:

None



4.5 Select Query, Joins and subquery

Using Comparison Operators - Examples

- Example 1: To display all staff details of who earn salary least salary

```
SELECT staff_name, staff_code, staff_sal
    FROM staff_master
      WHERE staff_sal  = (SELECT MIN(staff_sal)
                          FROM staff_master) ;
```

- Example 2: To display staff details who earn salary greater than average salary earned in dept 10

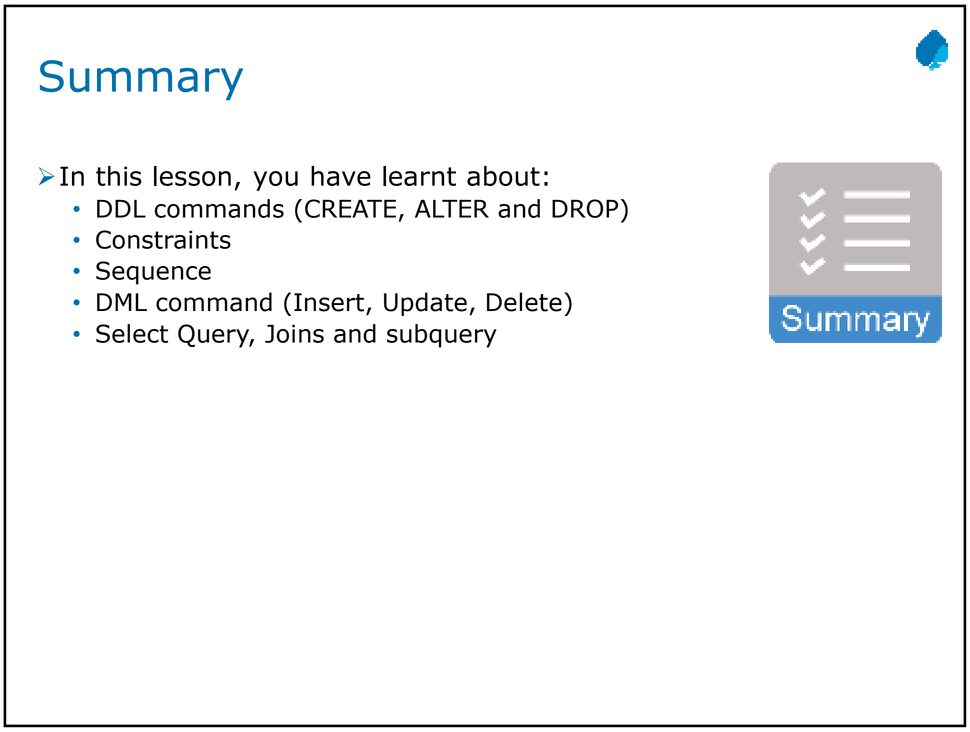
```
    SELECT staff_code,staff_sal
         FROM staff_master
         WHERE staff_sal > ANY(SELECT AVG(staff_sal)
                               FROM staff_master  GROUP BY
dept_code);
```

- For Single Row Subquery the sub-query should result in one value of the same data type as the left-hand side.
- Similarly for Multiple Row Subquery the list of values generated by the sub-query should be of same data type as the left-hand side
- The slide above shows examples of Subqueries. The first query is an example of Single Row Subquery and the second is an example of Multiple row Subquery

Instructor Notes:

Case Study

Instructor Notes:



Summary

- ➢In this lesson, you have learnt about:
  - DDL commands (CREATE, ALTER and DROP)
  - Constraints
  - Sequence
  - DML command (Insert, Update, Delete)
  - Select Query, Joins and subquery

Summary

Instructor Notes:

Question 1: A,B,C
Question 2: B

Review Question

- Question 1: If a sub-query returns multiple values, then the valid operators is/are ___.
  - Option 1: =
  - Option 2: IN
  - Option 3: >
  - Option 4: Any

- Question 2: A sub-query can be used for creating and inserting records.
  - True / False

Instructor Notes:

Question 1: A,B,C
Question 2: B

Review Question

- Question 3 : Inserting rows in a table emp1 from another table can be done using ____.
  - Option 1: insert into emp1(t1) as select empno from emp
  - Option 2: insert into emp1(t1) select empno from emp
  - Option 3: insert into emp1(t1) as select * from emp

- Question 4: Both TRUNCATE statement and DELETE without condition removes the entire date from a table
  - True/False

Instructor Notes:

Question 1: A,B,C
Question 2: B

Review Question

- Question 5 : ____________ join is based on any other operator other than equality
- Question 6: _____________ join joins the table to itself
- Question 7: ________ join includes a "+" operator along with equality operator