***Lesson 5***

**Column Aliases: -** we used to Renames a column heading

**Example 1**: select first\_name as name from employees;

**Example 2**: select first\_name "First Name" from employees;

***Lesson 6***

**Concatenation Operator: -**

* Links columns or character strings to other columns.
* Is represented by two vertical bars (||).
* Creates a resultant column that is a character expression.

**Example 1**: select last\_name||job\_id as "Employees" from employees;

**Example 2**: select last\_name||' '||job\_id as "Employees" from employees;

**Alternative Quote (q) Operator:**

* Specify your own quotation mark delimiter.
* Select any delimiter.
* Increase readability and usability.

**Example**: select last\_name||q'( job's id is: )'||job\_id as "Employees" from employees;

**Duplicate Rows:** removes duplicates from the table.

**Example**: select distinct DEPARTMENT\_ID from employees;

**Describe:** describe the created tables.

**Example**: desc employees;

***Lesson 7***

**Restricting & Sorting data:** using where to restrict or sort the data

**Example**: SELECT employee\_id, last\_name, job\_id, department\_id, salary, hire\_date

FROM employees

WHERE hire\_date = '17-feb-04';

**Comparison Operator:** =, >, >=, <, <=, <>, BETWEEN…...AND…., IN (SET, SET2), LIKE, IS NULL

**Logical Operator:**

* AND: Return true if both are true
* OR: Return true if either is true
* NOT: Return true if false

**Rules of AND & OR:** Oracle always take AND first so it’s better to use () to get what actually you need.

**Example:**  SELECT employee\_id, last\_name, job\_id, department\_id, salary, hire\_date

from employees

where (job\_id = 'SA\_REF'

OR job\_id = 'AD\_PRES')

AND salary > 10000;

***Lesson 8***

**Order By:** its always in the last line of the code, we can us any order can be in the table with **ASC & DESC** and we can use 2 order and use one DESC and one ASC.

**Example:** SELECT first\_name, last\_name, employee\_id, hire\_date

FROM employees

ORDER BY hire\_date, first\_name desc;

***Lesson 9***

**Substitution Variables:** We used to make a input pop up for user to search about what he needs, and we can use **define Variables** to define the variables in the first of the code and make it usable.

**Example:** define col1 = salary

SELECT first\_name, last\_name, employee\_id, hire\_date, &col1

FROM employees

WHERE &col1 > 12000

ORDER BY &col1;

undefine col1

***Lesson 10***

**Single Row Functions:**

* Character
* Number
* Data
* Conversion
* General

**Character:**

* Case-Conversion Function
* Character-Manipulation Function

**Case-Conversion Function:**

* LOWER: Change the character from upper to lower letters
* UPPER: Change the character from lower to upper letters
* INITCAP: Change the first letter to upper

**Example:** SELECT UPPER(first\_name) "LAST NAME", LOWER(last\_name) "first name", INITCAP(job\_id) "Job id"

from employees;

***Lesson 11***

**Character-Manipulation Function:**

* **CONCAT**: it's not usable and it’s the same as Concatenation Operation || is more advanced.
  + SELECT first\_name||last\_name, CONCAT(first\_name, last\_name) FROM employees;
* **SUBSTR**: it’s used to select several letters from a word.
  + select first\_name, last\_name, job\_id FROM employees WHERE SUBSTR(job\_id,1,2) = 'SA';
* **LENGTH:** It’s used to know the number of the letters in the word.
  + SELECT first\_name, last\_name, LENGTH(first\_name) FROM employees;
* **INSTR:** it’s used to know the position number in the word.
  + SELECT first\_name, last\_name, INSTR(first\_name, 'n',1,2) FROM employees;
* **LPAD:** It’s used to add letters to the LEFT of the word.
  + SELECT first\_name, last\_name, LPAD(salary, LENGTH(SALARY)+1, '$') FROM employees;
* **RPAD:** It’s used to add letters to the RIGHT of the word.
  + SELECT first\_name, last\_name, RPAD(salary, LENGTH(SALARY)+1, '$') FROM employees;
* **REPLACE:** It’s used to REPLACE letters with other letters.
  + SELECT first\_name, last\_name, REPLACE(first\_name, 'e', 'i') FROM employees;
* **TRIM:** It’s used to remove the beginning of the word and the end of it.
  + SELECT TRIM(' ' FROM ' Nader Mamdouh ') FROM dual;

***Lesson 12***

**Number Functions:**

* **ROUND:** Rounds value to a specified decimal.
  + Example: Select ROUND(63.323,2), ROUND(63.323,0), ROUND(63.323,-1), ROUND(63.323,-2), ROUND(446.323,-3) FROM dual;
* **TRUNC**: Truncates value to a specified decimal.
  + Example: Select TRUNC(63.326,2), TRUNC(63.326, 0), TRUNC(93.326,-2) FROM dual;
* **MOD**: Returns the remainder of the division.
  + Example: Select MOD(16, 2), MOD(17, 2) FROM dual;

***Lesson 13***

**Oracle DATE:** The Oracle Database stores dates in numbers function that we can use in ARITHMETIC ways,

* **format**: century, year, month, day, hours, minutes, and seconds.
* **The default** date display format is DD-MON-RR.
* **SYSDATE**: is a function that returns Date and Time from the Database server.
  + Example: SELECT SYSDATE FROM dual;
* **CURRENT\_DATE**: is a function that returns Date and Time from the client.
  + Example: SELECT CURRENT\_DATE FROM dual;

**Date-Manipulation Functions:**

* **MONTH\_BETWEEN:** Number of months between two dates.
  + Example: SELECT first\_name, hire\_date, ROUND(MONTHS\_BETWEEN(sysdate, hire\_date )/12, 1) FROM employees;
* **ADD\_MONTHS:** Add calendar months to date.
  + Example: SELECT first\_name, hire\_date, ADD\_MONTHS(hire\_date,1) FROM employees;
* **NEXT\_DAY:** Next day of the date specified.
  + Example: SELECT first\_name, sysdate, NEXT\_DAY(SYSDATE, 'MON') FROM employees;
* **LAST\_DAY:** Last day of the month of the date specified.
  + Example: SELECT first\_name, hire\_date, LAST\_DAY(HIRE\_DATE) FROM employees;
* **ROUND:** Round date.
  + Example: SELECT first\_name, hire\_date, ROUND(HIRE\_DATE, 'MONTH'), ROUND(HIRE\_DATE, 'YEAR') FROM employees;
* **TRUNC:** Truncate date.
  + Example: SELECT first\_name, hire\_date, TRUNC(HIRE\_DATE, 'MONTH'), TRUNC(HIRE\_DATE, 'YEAR') FROM employees;

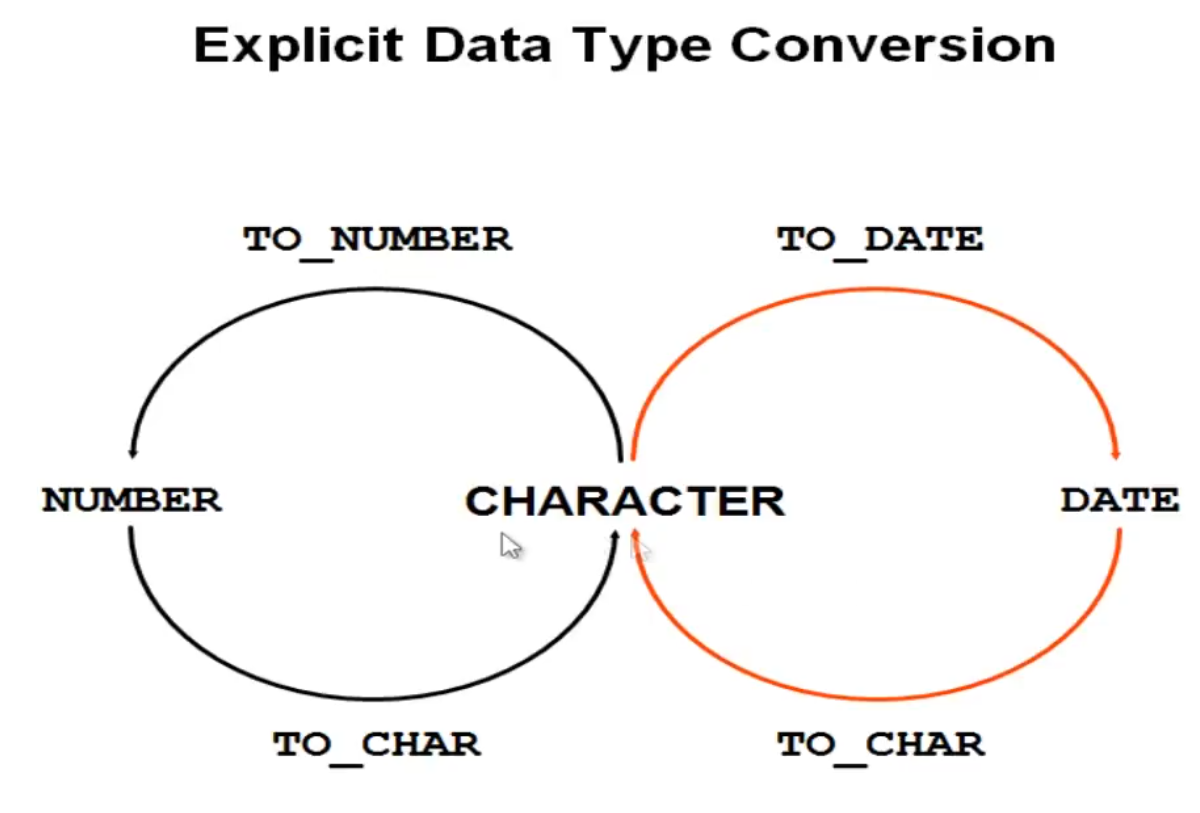
***LESSON 14***

**Conversion Function:**

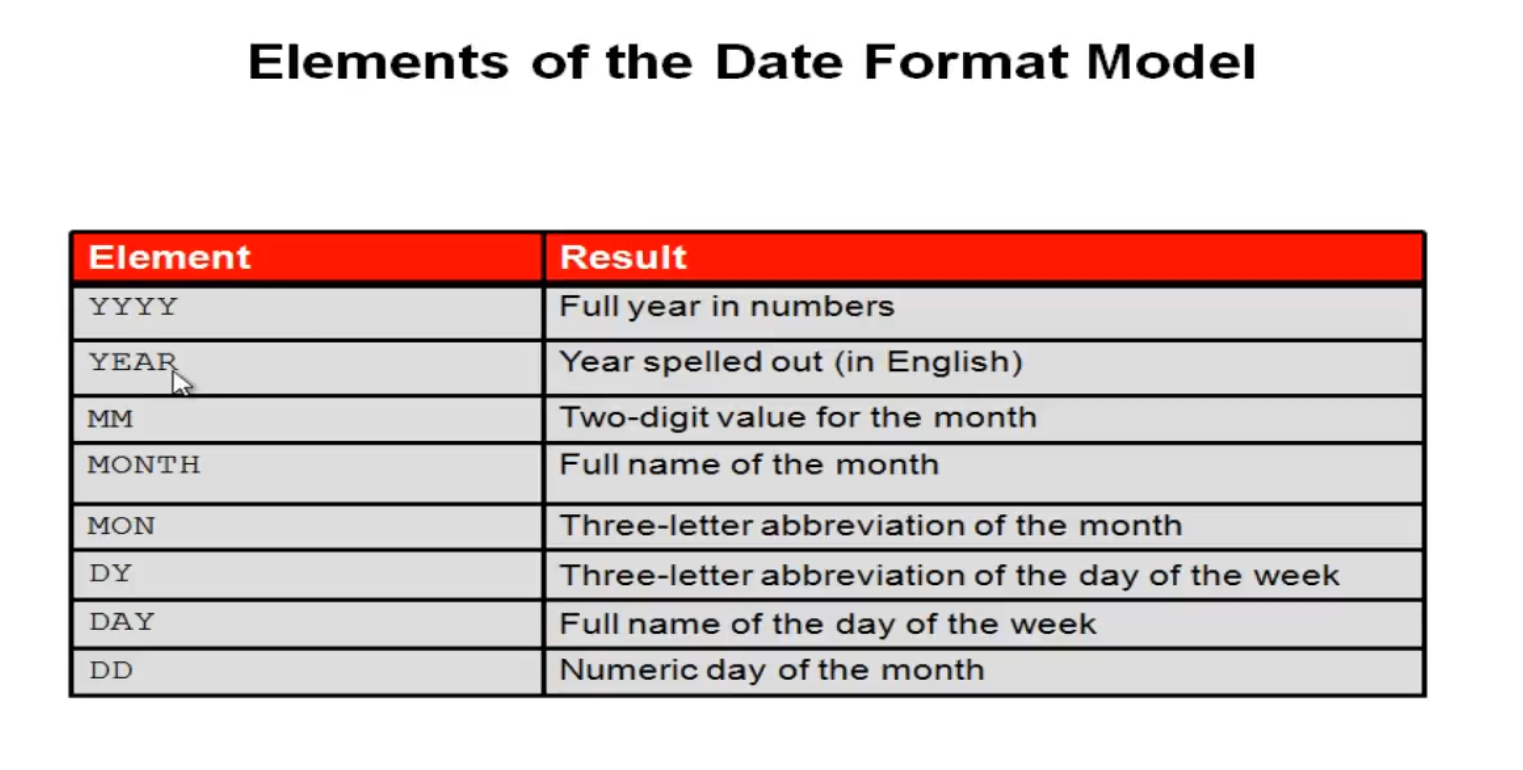
**Data Type Conversion: -**

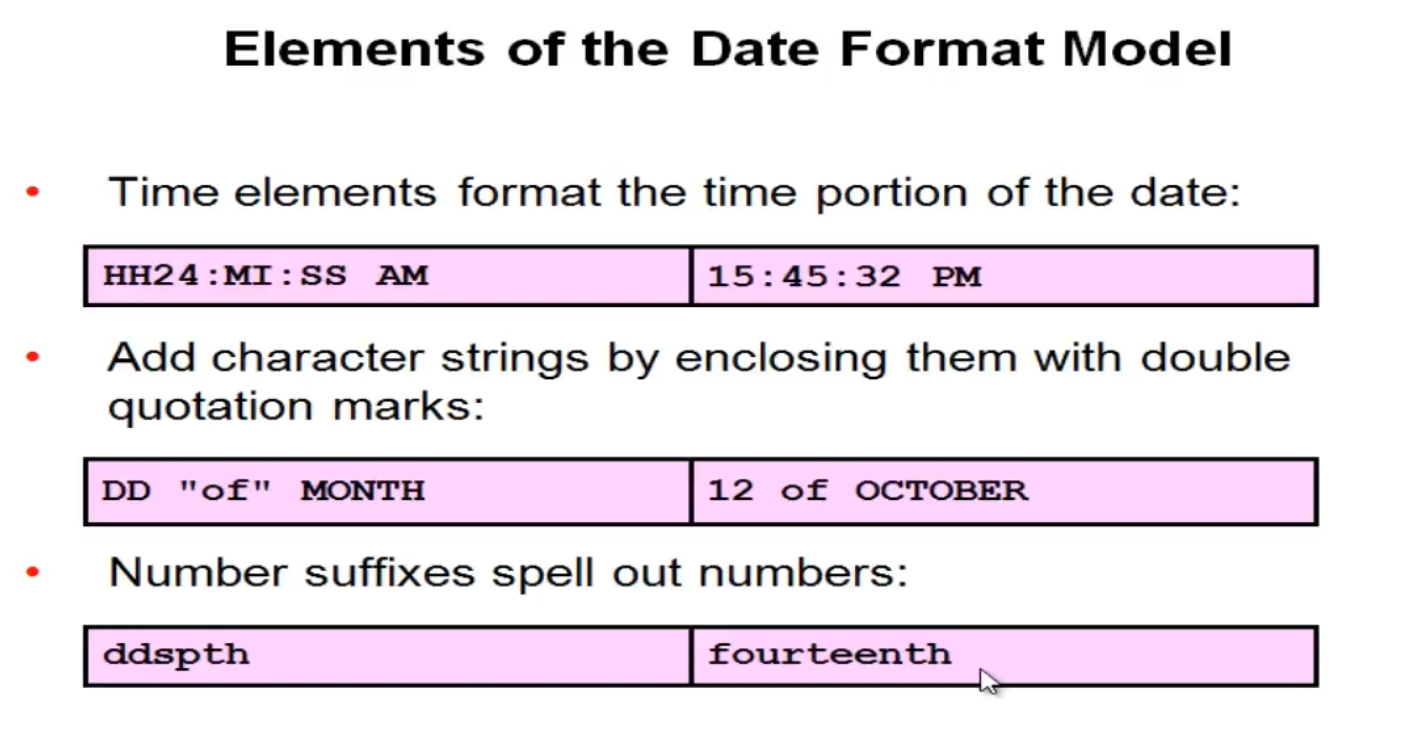
* **Implicit data type conversion:** In expressions, the Oracle Server can automatically convert the following:
  + VARCHAR2 or CHAR to NUMBER or DATE
    - Example: SELECT last\_name, employee\_id FROM employees WHERE employee\_id = '200';

**Explicit data type conversion:** this uses functions to change the data type from number or date to char or the opposite:

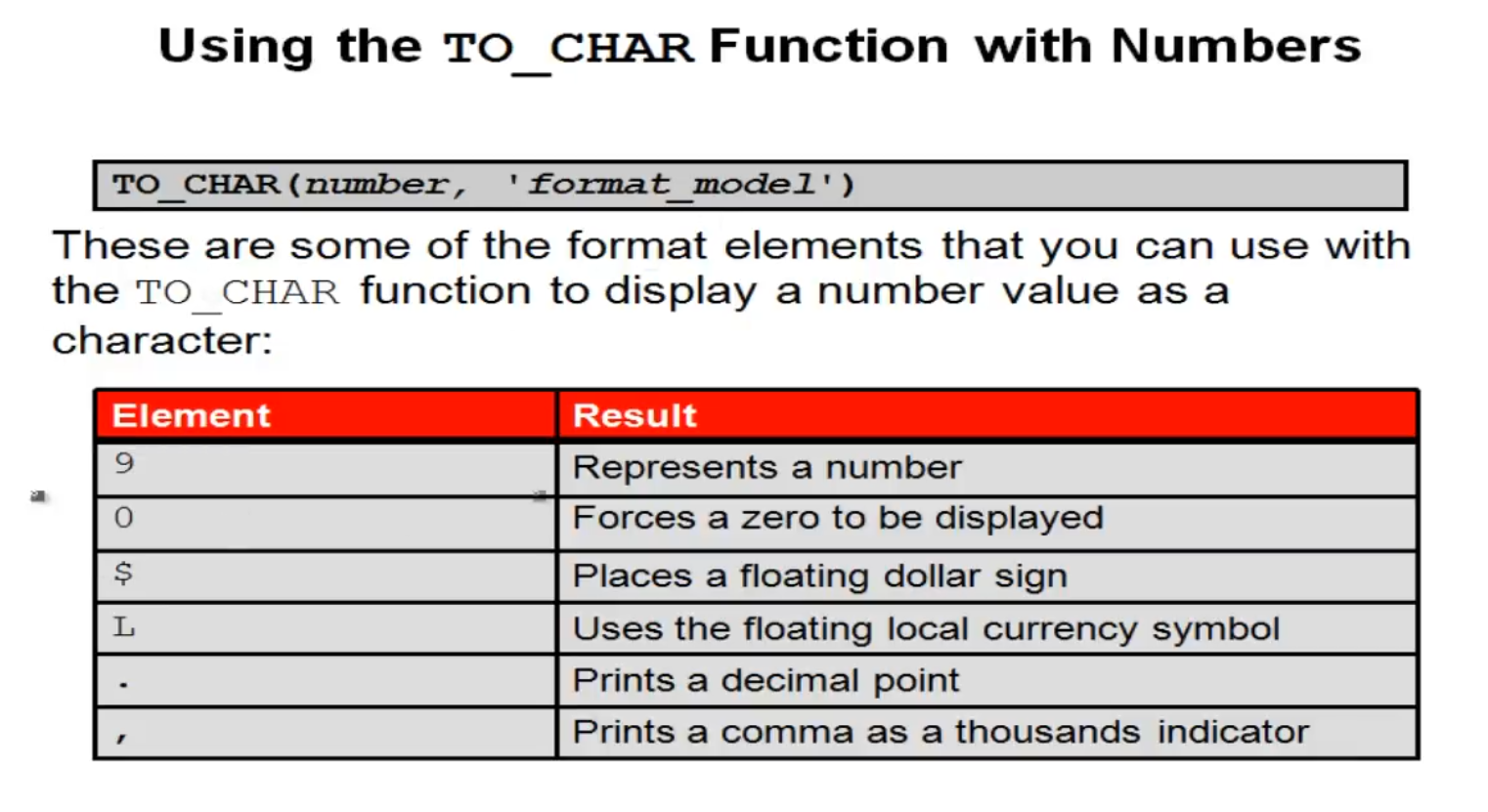


* **TO\_CHAR:** Used to change date or number to char.
  + Example1: SELECT hire\_date, TO\_CHAR(hire\_date, 'dd-mm-yyyy') FROM employees;

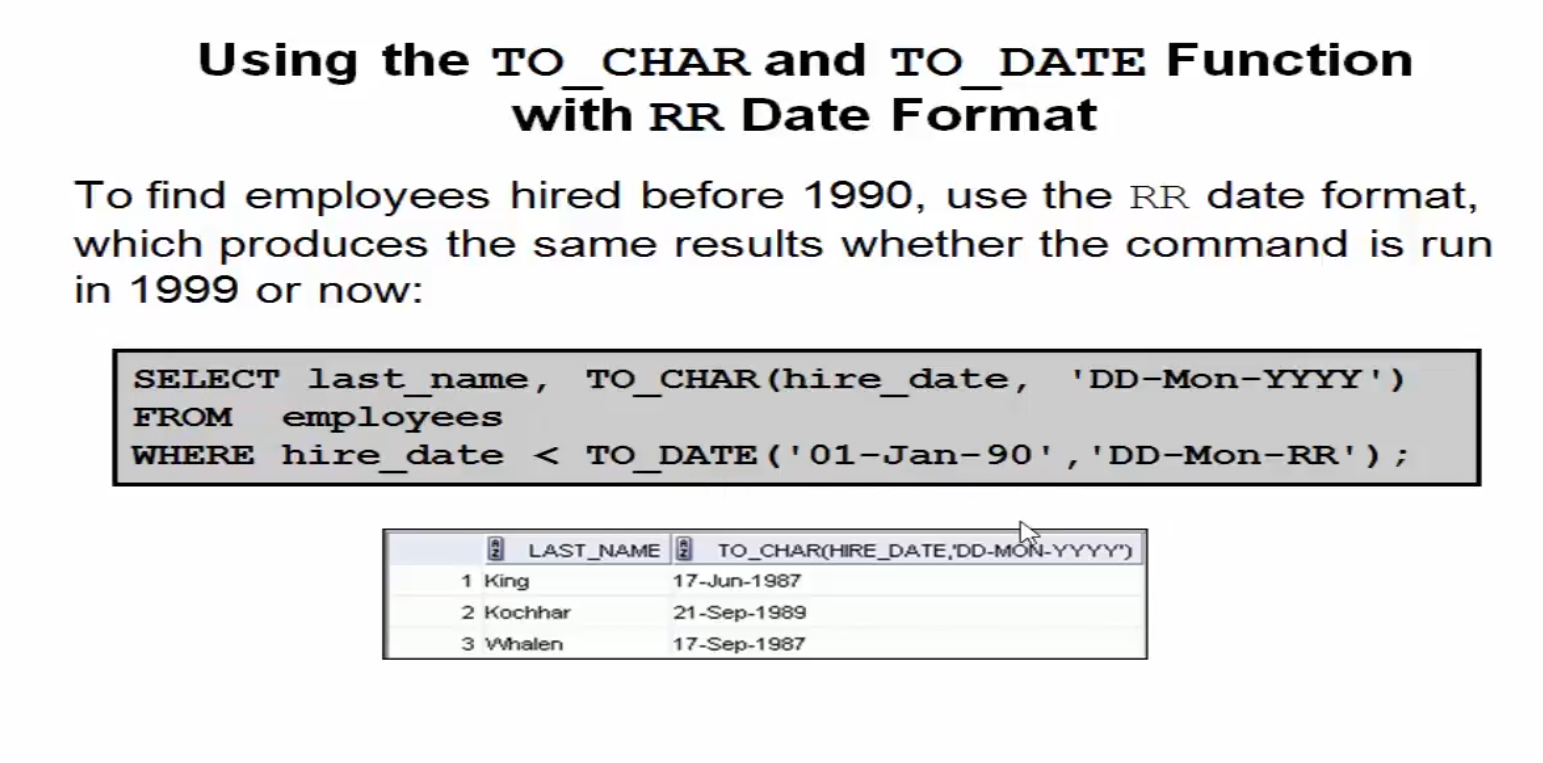




* + Example2: SELECT first\_name, salary, TO\_CHAR(salary, 'L999,999,999.00') FROM employees;



* **TO\_DATE:** Convert a character string to a DATE format**.**
  + Example1: SELECT last\_name, hire\_date FROM employees WHERE hire\_date <= TO\_DATE('01/02/2007', 'dd-mm-yyyy');



* **TO\_NUMBER**: Convert a character string to a number format.
  + Example1: SELECT TO\_NUMBER('01112008903', '99999999999999') FROM dual;

***LESSON 15***

**Nesting Functions:**

* Single-Row Function can be nested to any level.
* Nesting Functions are evaluated from the deepest level to the least deep level.
  + Example: SELECT first\_name, UPPER(CONCAT(SUBSTR(first\_name, 1, 3), '\_MI')) FROM employees;

**General Functions:** The following functions work with any data type and pertain to using null.

* **NVL**: Changing NULL to numbers to use it in mathematics.
  + Example: SELECT first\_name, salary, commission\_pct, NVL(commission\_pct, 1) FROM employees;
* **NVL2**: This means if null, give a result and we can make the result be char.
  + Example: SELECT first\_name, salary, commission\_pct, nvl2(commission\_pct, salary\*1.1, 0) FROM employees;
* **NULLIF**: Compare between 2 values and if == give you null and if \= result value 1.
  + Example: SELECT length(first\_name), length(last\_name), NULLIF(length(first\_name), length(last\_name)) FROM employees;
* **COALESCE**: search between values and when find the first not null value get it.
  + Example: SELECT COALESCE(NULL, NULL, NULL, 1,NULL) FROM dual;

**Conditional Expressions:**

* **CASE:**
  + Example: SELECT last\_name, job\_id , salary,

CASE job\_id WHEN 'PU\_MAN' THEN 1.15 \* salary

WHEN 'SH\_CLERK' THEN 1.3 \* salary

WHEN 'AC\_ACCOUNT' THEN 1.4 \* salary

WHEN 'FI\_ACCOUNT' THEN 1.51 \* salary

ELSE SALARY \* 1.1 END "Annual Salary increase" FROM employees;

* **DECODE:** 
  + Example: last\_name, job\_id , salary,

DECODE( job\_id , 'PU\_MAN' , 1.15 \* salary

, 'SH\_CLERK' , 1.3 \* salary

, 'AC\_ACCOUNT' , 1.4 \* salary

, 'FI\_ACCOUNT' , 1.51 \* salary

, SALARY \* 1.1) "Annual Salary increase" FROM employees;

***LESSON 16***

**Group Functions:** Used in a group of rows to get the Output and we can use null and distinct.

* **AVG:** Get the Average for a group of numbers.
  + Example: SELECT AVG(salary) "Average" FROM employees;
* **COUNT:** Count the number of rows in columns.
  + Example: SELECT COUNT(distinct salary) "SUM" FROM employees;
* **MAX:** Get the higher number the newest date or the last letter.
  + Example: SELECT MAX(salary), MAX(hire\_date), MAX(first\_name) "SUM" FROM employees;
* **MIN:** Get the smallest number the oldest date or the first letter.
  + Example: SELECT MIN(salary), MIN(hire\_date), MIN(first\_name) "SUM" FROM employees;
* **SUM:** get the sum for a group of numbers.
  + Example: SELECT SUM(salary) "SUM" FROM employees;
* **STDDEV**
  + Example:
* **VRINANCE**
  + Example:

***LESSON 17***

**BROUP BY:** You can divide rows in a table into smaller groups by using the GROUB BY clause.

* Example: SELECT department\_id, job\_id, commission\_pct, count(\*) FROM employees

WHERE department\_id = 50

GROUP BY department\_id, job\_id, commission\_pct ORDERBY 2;

**HAVING:** we used likes order by but with GROUP BY.

* Example: SELECT department\_id, job\_id, commission\_pct, count(\*) FROM employees

WHERE department\_id = 50

GROUP BY department\_id, job\_id, commission\_pct

HAVING count(\*) > 5 ORDER BY 2;

**NESTING GROUP FUNCTIONS:** we can use nesting with GROUP FUNCTIONS

* Example: SELECT MAX(avg(salary)) FROM employees GROUP BY salary;

***LESSON 18***

**Types Of Joins**: Joins that are compliant with the SQL: 1999 standard include the following:

* **NATRUAL JOINS:** Is 1 of the Joins types
  + **NATRUAL JOINS**: We Used to join 2 tables with the PK and FK its have to be the same data type
    - Example: SELECT department\_id, department\_name, location\_id, city FROM departments d NATURAL JOIN locations;
  + **USING**: we used it when we needed to join 2 tables with different data types.
    - Example: SELECT department\_id, department\_name, location\_id, city FROM departments d JOIN locations USING (location\_id);
  + **ON**: We used when we needed to join 2 tables with different names.
    - Example: SELECT department\_id, department\_name location\_id, city FROM departments d JOIN locations l ON (d.location\_id = l.location\_id);
  + **More than 2 tables**: we can use more than 2 tables.
    - Example: SELECT e.last\_name, d.department\_id, department\_name, l.location\_id, city FROM departments d JOIN locations l ON (d.location\_id = l.location\_id) JOIN employees e ON d.manager\_id = e.employee\_id;
  + **WHERE & AND & OR:** we can use where, and, or in different table.
    - **Example:** SELECT e.last\_name, d.department\_id, department\_name, l.location\_id, city FROM departments d JOIN locations l ON (d.location\_id = l.location\_id) JOIN employees e ON d.manager\_id = e.employee\_id WHERE d.department\_id = 70 OR d.department\_name = 'IT' AND e.last\_name = 'Hunold';
  + **SELF JOIN:** You can make 2 tables and join each other from the table itself.
    - **Example:** SELECT a.employee\_id worker, a.last\_name, b.employee\_id manager, b.last\_name FROM employees a JOIN employees b ON a.manager\_id = b.employee\_id;
  + **None EquiJoins:** Retrieving Records with NoneEquiJoins.
    - **Example:** SELECT e.last\_name, e.salary, j.grade\_level FROM employees e JOIN job\_grades j ON e.salary BETWEEN j.lowest\_sal AND j.highest\_sal;

***LESSON 19***

* **OUTER JOIN:**
  + **LEFT OUTER JOIN:** shows the nulls value of the left columns.
    - **Example:** SELECT a.employee\_id worker, a.last\_name, b.employee\_id manager, b.last\_name FROM employees a LEFT OUTER JOIN employees b ON a.manager\_id = b.employee\_id;

* + **RIGHT OUTER JOIN:** shows the nulls value of the right columns.
    - **Example:** SELECT e.last\_name, d.department\_id, d.department\_name FROM employees e RIGHT OUTER JOIN departments d ON (e.department\_id = d.department\_id);
  + **FULL OUTER JOIN:** shows the nulls value of ALL columns.
    - **Example:** SELECT e.last\_name, d.department\_id, d.department\_name FROM employees e FULL OUTER JOIN departments d ON (e.department\_id = d.department\_id);

* **Cartesian Product:** All rows in the first table are joined to all rows in the second table, to avoid a Cartesian product, always include a valid join condition.
  + **CROSS JOIN:** 
    - SELECT last\_name, department\_name FROM employees CROSS JOIN departments;

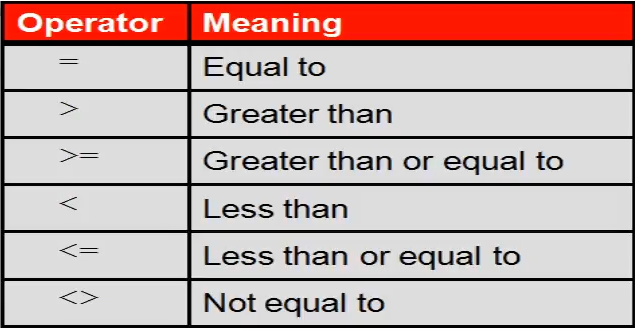
***LESSON 20***

**Subquery:** We can used in make another select statement to get a specific output to used in the main statement.

* we use it in ().
* Its better to put it in the right the code to easy read.
* We use Single-row operators with Single-row subquery and the Multiple-row operators with Multiple-row subquery

**Example**: SELECT last\_name, salary FROM employees WHERE salary > (SELECT salary FROM employees WHERE last\_name = 'Abel');

**Single-row subqueries**: Return only one row & use single- row comparison operators.



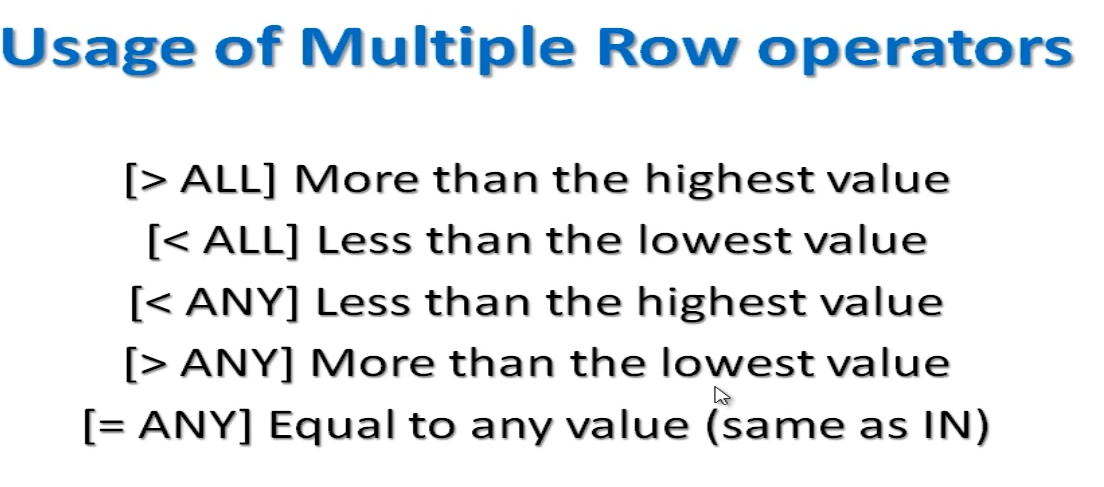
**Example:** SELECT last\_name, salary FROM employees WHERE salary > (SELECT salary FROM employees WHERE last\_name = 'Abel') AND salary < (SELECT salary FROM employees WHERE last\_name = 'King' and salary > 10000);

* **Group Functions in a Subquery:** we can use group functions in and with subqueries.
  + **Example:** SELECT last\_name, salary FROM employees WHERE salary > (SELECT AVG(salary) "Average" FROM employees) AND salary < (SELECT MAX(salary) "MAX" FROM employees);
* **HAVING clause with Subquery:**
  + **Example:** SELECT department\_id, min(salary) FROM employees GROUP BY department\_id HAVING min(salary) > (SELECT min(salary) "Minimum" FROM employees WHERE department\_id =50);

***LESSON 21***

**Multiple-row subqueries:** when we need to use multiple row to used.

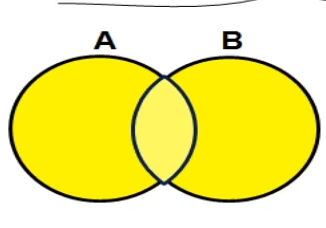
* **IN :** SELECT employee\_id, last\_name, job\_id, salary FROM employees WHERE salary IN OR = ANY (SELECT salary FROM employees WHERE job\_id = 'IT\_PROG') AND job\_id <> 'IT\_PROG';
* **ANY :** SELECT employee\_id, last\_name, job\_id, salary FROM employees WHERE salary < ANY (SELECT salary FROM employees WHERE job\_id = 'IT\_PROG') AND job\_id <> 'IT\_PROG';
* **ALL :** SELECT employee\_id, last\_name, job\_id, salary FROM employees WHERE salary < ALL (SELECT salary FROM employees WHERE job\_id = 'IT\_PROG') AND job\_id <> 'IT\_PROG';
* **NULL VALUES :** will always get 0 rows if we have null;
  + **Example:** SELECT emp.last\_name FROM employees emp WHERE emp.employee\_id NOT IN (SELECT mgr.manager\_id FROM employees mgr);



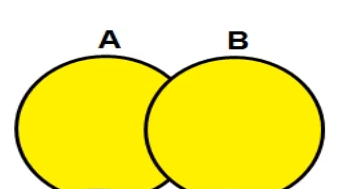
***LESSON 22***

**Set Operators:** When we need to get the data from 2 tables or the different from this data.

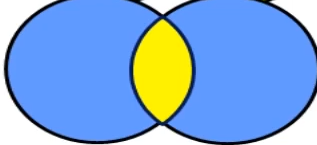
* **UNION ALL:** get the all data from the 2 tables with the duplicate
  + **Example:** SELECT employee\_id, job\_id FROM employees WHERE employee\_id = 200 UNION ALL SELECT employee\_id, job\_id FROM job\_history WHERE employee\_id = 200;



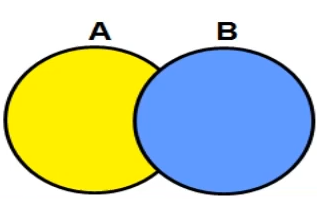
* **UNION:** get the all data from the 2 tables without the duplicate
  + **Example:** SELECT employee\_id, job\_id FROM employees WHERE employee\_id = 200 UNION SELECT employee\_id, job\_id FROM job\_history WHERE employee\_id = 200;



* **INTERSECT:** Get the duplicated area from the 2 tables
  + **Example:** SELECT employee\_id, job\_id FROM employees WHERE employee\_id = 200 INTERSECT SELECT employee\_id, job\_id FROM job\_history WHERE employee\_id = 200;



* **MINUS:** Get the data from the first table that isn’t in the second table” we can say from the example that we can’t the data that didn’t changed”.
  + **Example:** SELECT employee\_id, job\_id FROM employees MINUS SELECT employee\_id, job\_id FROM job\_history;



* **MATCHING:** that is a rule that we can’t get more info from the first table without matching the output from the second table.
  + **Example:** SELECT employee\_id, job\_id, salary, 'current' FROM employees UNION SELECT employee\_id, job\_id, 0, to\_char(end\_date, 'dd/mm/yyyy') FROM job\_history;

***LESSON 23***

**DML** : Data Manipulation Language.

* A DML Statement is executed when you:
  + Add new rows to a table.
  + Modify existing rows in a table.
  + Remove existing rows from a table.
* A transaction consists of a collection of DML statements that from a logical unit of work.

Adding New row to a Table:

* **INSERT Statement Syntax:** Add new row to a table by using INSERT statement 1 row at a time.
  + **Example:** INSERT INTO departments(department\_id, department\_name, manager\_id, location\_id) VALUES (300, 'IT\_NADER', 200, 1700);
* **INSERT rows with null values:** we have 2 methods.
  + **Implicit method:** omit the column from the column list.
    - **Example:** INSERT INTO departments(department\_id, department\_name) VALUES (301, 'IT\_NADER');
  + **Explicit method:** Specify the NULL keywords in the VALUES.
    - **Example:** INSERT INTO departments(department\_id, department\_name, manager\_id, location\_id) VALUES (302, 'IT\_NADER', NULL, NULL);
* **INSERT Special and Specific Values:** The SYSDATE function records the current date and time.
  + **Example 1:** INSERT INTO employees(employee\_id, last\_name, email, hire\_date, salary, commission\_pct, job\_id) VALUES (300, 'Nader', 'nader', SYSDATE, 2000, 0.2, 'IT\_PROG');
  + **Example 2:** INSERT INTO employees(employee\_id, last\_name, email, hire\_date, salary, commission\_pct, job\_id) VALUES (301, 'Mamdouh', 'nader1', to\_date('31/10/2024', 'dd/mm/yyyy'), 2001, 0.2, 'IT\_PROG');
* **Creating a Script:** Use & substitution in a SQL statement to prompt for values (& is a placeholder for the variable value.).
  + **Example 1:** INSERT INTO departments(department\_id, department\_name) VALUES (&D\_ID, '&D\_Name');
* **Copying Rows from Another Table:** Write your INSERT statement with a subquery( you have to match the numbers of columns in the INSERT clause in the subquery).
  + **Example 1:** INSERT INTO NEW\_EMPLOYEES(id, first\_name, last\_name, salary, commission\_pct) SELECT employee\_id, first\_name, last\_name, salary, commission\_pct FROM employees WHERE job\_id = 'IT\_PROG';

***LESSON 24***

Thats code to create a copy of Employees table:

CREATE TABLE COPY\_EMPLOYEES AS SELECT \* FROM EMPLOYEES;

**UPDATE:** we use it to update data in table and we have to use WHERE because if we didn’t that will update the entire table.

* **Example 1:** update copy\_employees set department\_id = 90;
  + in example 1 we use it without WHERE and the entire table updated the department\_id to 90.
* **Example 2:** update copy\_employees set department\_id = 90 WHERE employee\_id = 113;

**UPDATE with subquery:** We can use subquery to update row.

* **Example:** update copy\_employees set salary = (select salary from employees where employee\_id = 206), department\_id = (select department\_id from employees where employee\_id = 206) WHERE employee\_id = 113;

**And we can use null.**

* **Example:** update copy\_employees set salary = null WHERE employee\_id = 113;

***LESSON 25***

Note: CREATE TABLE employees\_copy AS SELECT \* from employees

**Removing a row from a table:** We can delete a row from a table with the syntax DELETE.

* **Example:** DELETE employees\_copy WHERE employee\_id = 105;

**We can use Subquery with delete:**

* **Example:** DELETE employees\_copy WHERE employee\_id = (SELECT employee\_id FROM EMPLOYEES WHERE SALARY > 20000);

When we remove WHERE we will delete all the table data and if we did we can use ROLLBACK to fix it.

**Example:** DELETE employees\_copy; ROLLBACK;

**TRUNCATE:** this remove the all table or make a delete but we cant comeback after DDL langauge.

**Example:** TRUNCATE employees\_copy

***LESSON 26***

**Database Transactions:** Start and End

* Begin when the DML SQL Statement is executed.
* End with one of the following events:
* A COMMIT or ROLLBACK Statement is issued.
* A DDL or DCL Statement executes (Automatic commit).
* The User exit SQL Developer or SQL plus.
* The system crashes.

**COMMIT and ROLLBACK:** its like a savepoints and we can use it to be sure and before make the changes.

**ROLLBACK:** its make you back from the begining and remove the all changes you did it

* **Example:** DELETE employees\_copy; ROLLBACK;

**COMMIT:** its commit all changes and save the data.

* **Example:** DELETE employees\_copy; COMMIT;

**SAVEPOINT:** its saveing changes and use it to back to savepoint we choose.

* **Example:** UPDATE employees\_copy SET SALARY = 10000 WHERE employee\_id = 103;

SAVEPOINT a;

UPDATE employees\_copy SET SALARY = 10000 WHERE employee\_id = 104;

SAVEPOINT b;

UPDATE employees\_copy SET SALARY = 10000 WHERE employee\_id = 105;

DELETE employees\_copy;

ROLLBACK TO SAVEPOINT b;

**FOR UPDATE:** use to lock the other users from select the rows you use and we can make from many tables so we can use joins.

* **Example:** SELECT e.employee\_id, e.salary, e.commission\_pct FROM EMPLOYEES e JOIN departments d USING (department\_id)

WHERE job\_id = 'ST\_CLERK' AND location\_id = 1500

FOR UPDATE

ORDER BY e.employee\_id;

***LESSON 27***

Notes: Referencing Another User’s Tables: We can use it if we make a user for performance and user for claim’s and user for management’s we can used to connect the tables with each other.

**Database Objects:**

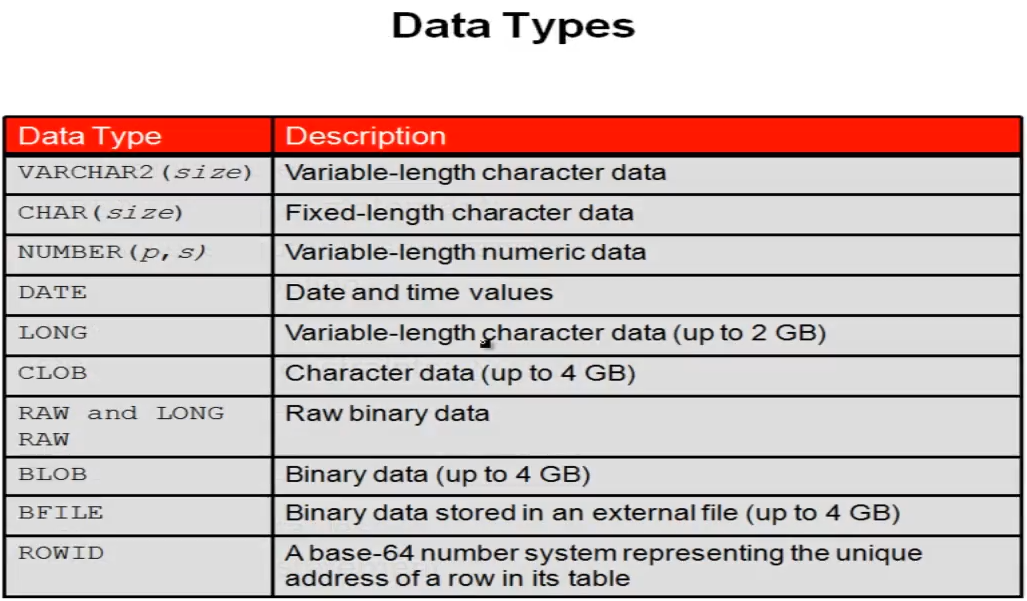
* **Table:** Basic Unit of storage, composed with rows
* **View:** Its show you data from tables to edit and represent.
* **Sequence:** Generates Numeric values.
* **Index:** Improves the performance of some queries.
* **Synonym:** we can use to give sample names to objects.

**Naming Rules:**

* Must to begin with a latter.
* Must be 1-30 char long.
* Must be contain only A-Z, a-z, 0-9, \_, $, and #.
* Must not duplicate the object name in the same user.
* Most not be an Oracle server words like: SELECT, FROM, ……etc.

**CREATE TABLE Statement:**

* **Example:** CREATE TABLE DEMP(d\_id number(9), first\_name varchar2(30), last\_name varchar2(30), create\_date date DEFAULT sysdate);



***LESSON 28***

**Constraints:** Is to make rules make less miss entering wrong data.

* **NOT NULL:** Can’t make the table null or empty.
* **UNIQUE:** Is a unique data can be duplicated.
* **CHECK:** Check if the data entered is required.
* **PRIMARY KEY:** It’s a unique key generated and cant be duplicated.
* **FOREIGN KEY:** It's make reference to PK or another table and can be duplicated and can be null.
  + **Example:** CREATE TABLE EMP(

emp\_id number(10),

emp\_name varchar2(30) NOT NULL,

salary NUMBER(8,2),

email varchar2(50),

dept\_id number(10),

CONSTRAINT emp\_pk PRIMARY KEY(emp\_id),

CONSTRAINT sa\_ck CHECK(salary > 1000) ,

CONSTRAINT em\_un UNIQUE(email),

CONSTRAINT dept\_fk FOREIGN KEY(dept\_id) REFERENCES DEPARTMENTS(DEPARTMENT\_ID));

***LESSON 29***

**Create Table Using Subquery:** We can copy a table with subquery from another table.

**Example:** CREATE TABLE EMP100 AS (SELECT \* FROM employees WHERE department\_id = 100