

A thick dark blue vertical bar runs along the left edge of the slide. A blue arrow-shaped banner points to the right from this bar, containing the date. In the bottom-left corner, several thin, curved lines in dark blue and light grey sweep upwards and to the right.

17-10-2022

# Database Programming with SQL 2 / 2

JUAN CARLOS HERRERA HERNANDEZ  
UNIVERSIDAD POLITECNICA DE AGUASCALIENTES

## Contenido

Section 11 – Ensuring Quality Queries Part I .....	2
11-1 Ensuring Quality Query Results .....	2
Section 12 – DML .....	2
12-1 INSERT Statements .....	2
12-2 Updating Column Values and Deleting Rows .....	3
12-3 DEFAULT Values, MERGE, and Multi-Table Inserts.....	4
Section 13 – DDL .....	5
13-1 Creating Tables .....	5
13-2 Using Data Types.....	6
13-3 Modifying a Table .....	7
Section 14 – Constraints .....	8
14-1 Intro to Constraints; NOT NULL and UNIQUE Constraints.....	8
14-2 PRIMARY KEY, FOREIGN KEY, and CHECK Constraints .....	9
14-3 Managing Constraints .....	10
Section 15 – Views .....	11
15-1 Creating Views .....	11
15-2 DML Operations and Views .....	12
15-3 Managing Views.....	12
Section 16 – Sequences and Synonyms .....	14
16-1 Working With Sequences.....	14
16-2 Indexes and Synonyms .....	15
Section 17 – Privileges and Regular Expressions .....	16
17-1 Controlling User Access .....	16
17-2 Creating and Revoking Object Privileges .....	17
17-3 Regular Expressions .....	19
Section 18 – TCL .....	20
18-1 Database Transactions.....	20

## Section 11 – Ensuring Quality Queries Part I

### 11-1 Ensuring Quality Query Results

Solve a series of problems:

- Solve a series of problems Create a query to produce specified data
- Modify a query to produce specified data

Select * from user_tables	PURGE RECYCLEBIN;
select * from tab;	

## Section 12 – DML

### 12-1 INSERT Statements

<b>USER</b>	Someone doing “real work” with the computer, using it as a means rather than an end
<b>Transaction</b>	Consists of a collection of DML statements that form a logical unit of work.
<b>Explicit</b>	Fully and clearly expressed; leaving nothing implied
<b>INSERT INTO</b>	Adds a new row to a table

The table copies will not inherit the associated primary-to-foreign-key integrity rules (relationship constraints) of the original tables.

Copy structure and data	Copy only structure
CREATE TABLE copy_departments as SELECT * FROM departments;	CREATE TABLE copy_departments as (SELECT * FROM departments where 1=2);

Describe employees	SALARY NUMBER(6,2) Precision 6, Scale 2 [-9999.99, 9999.99]
user	select user from dual;
sysdate	select sysdate from dual; -- default DD-Mon-YYYY
	select to_char(sysdate, 'Month fmdd, yyyy') from dual;

select columns	all columns
INSERT INTO copy_departments (department_id, department_name, location_id) VALUES (200,'Human Resources', 1500);	INSERT INTO copy_departments VALUES (210,'Estate Management', 102, 1700);

INSERT INTO copy_employees (employee_id, first_name, last_name, email, hire_date, salary) VALUES (302,'Grigorz','Polanski', 'GPolanski', TO_DATE('2017-07-20', 'yyyy-mm-dd'), 4200);
---

Insert multiple records at the same time
INSERT INTO sales_reps(id, name, salary, commission_pct) SELECT employee_id, last_name, salary, commission_pct FROM employees WHERE job_id LIKE '%REP%';

## 12-2 Updating Column Values and Deleting Rows

<b>UPDATE</b>	Modifies existing rows in a table
<b>Correlated subquery UPDATE</b>	retrieves information from one table & uses the information to update another table
<b>Integrity Constraint</b>	Ensures that the data adheres to a predefined set of rules
<b>Correlated subquery DELETE</b>	deletes information on a linked table based on what was deleted on the other table
<b>Delete</b>	Removes existing rows from a table

Not Correlated	Correlated
<pre>UPDATE copy_employees SET hire_date = sysdate WHERE employee_id = 206;</pre>	<pre>UPDATE copy_employees SET hire_date = sysdate,     salary = (SELECT salary FROM copy_employees               WHERE employee_id= 205),     job_id = (SELECT job_id FROM copy_employees               WHERE employee_id= 205) WHERE employee_id = 206;</pre>

Correlated
<pre>ALTER TABLE copy_employees ADD (department_name varchar2(30));  select e.department_id, d.department_id, d.department_name from employees e, departments d where e.department_id = d.department_id;  UPDATE copy_employees e SET e.department_name= (SELECT d.department_name                        FROM departments d                        WHERE e.department_id= d.department_id);</pre>

Not Correlated	Correlated
<pre>DELETE FROM departments WHERE department_id = 50;  DELETE FROM copy_employees WHERE department_id = 50;</pre>	<pre>DELETE FROM copy_employees WHERE department_id =     (SELECT department_id FROM departments      WHERE department_name= 'Shipping');</pre>

Be carefully	
<pre>SELECT * FROM copy_employees e WHERE e.manager_id IN     (SELECT d.manager_id      FROM employees d      GROUP BY d.manager_id      HAVING count(d.department_id) &lt; 2);</pre>	<pre>DELETE FROM copy_employees e WHERE e.manager_id IN     (SELECT d.manager_id      FROM employees d      GROUP BY d.manager_id      HAVING count(d.department_id) &lt; 2);</pre>

row-level locks, until you issue a COMMIT or ROLLBACK
<pre> SELECT e.employee_id, e.salary, d.department_name FROM employees e JOIN departments d USING (department_id) WHERE location_id = 1500 AND job_id= 'ST_CLERK' FOR UPDATE ORDER BY e.employee_id;  GRANT update, select ON employees TO schemas  User: SCHEMAS update ESQUEMAS.employees e set salary = salary where e.employee_id = 141; </pre>

### 12-3 DEFAULT Values, MERGE, and Multi-Table Inserts

A **data warehouse** is a collection of data designed to support business-management decision making. Data warehouses contain a wide variety of data, such as sales data, customer data, payroll, accounting, and personnel data, which presents a coherent picture of business conditions at a single point in time.

<pre> CREATE TABLE my_employees ( hire_date DATE DEFAULT SYSDATE, first_name VARCHAR2(15), last_name VARCHAR2(15)); </pre>	<pre> -- Explicit INSERT INTO my_employees (hire_date, first_name, last_name) VALUES (DEFAULT, 'Angelina','Wright'); </pre>	<pre> -- Implicit INSERT INTO my_employees (first_name, last_name) VALUES('Angelina','Wright'); </pre>
--	---	--

<pre> UPDATE my_employees SET hire_date = DEFAULT WHERE last_name = 'Wright'; </pre>	<pre> UPDATE my_employees SET hire_date = '21-SEP-89' WHERE last_name = 'Wright'; </pre>	<pre> UPDATE copy_employees SET hire_date = <b>to_date</b>('1989-09-21', 'yyyy-mm-dd') WHERE employee_id = 100; </pre>
--	--	--

<pre> MERGE will INSERT and UPDATE simultaneously.  MERGE INTO destination-table USING source-table ON matching-condition WHEN MATCHED THEN UPDATE SET ..... WHEN NOT MATCHED THEN INSERT VALUES (.....); </pre>	<pre> MERGE INTO copy_emp c USING employees e ON (c.employee_id = e.employee_id) WHEN MATCHED THEN UPDATE SET     c.last_name      = e.last_name,     c.department_id  = e.department_id WHEN NOT MATCHED THEN INSERT VALUES (e.employee_id, e.last_name, e.department_id); </pre>
--	--

# MERGE Example

EMPLOYEES (source table)

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
100	King	90
103	Hunold	60
142	Davies	50

COPY\_EMP before the MERGE is executed

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
100	Smith	40
103	Chang	30

COPY\_EMP after the MERGE has executed

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
100	King	90
103	Hunold	60
142	Davies	50

# Multi-Table Inserts Conditional

```

INSERT ALL
  WHEN call_format IN ('tlk','txt','pic') THEN
    INTO all_calls
      VALUES (caller_id, call_timestamp, call_duration, call_format)
  WHEN call_format IN ('tlk','txt') THEN
    INTO police_record_calls
      VALUES (caller_id, call_timestamp, recipient_caller)
  WHEN call_duration < 50 AND call_type = 'tlk' THEN
    INTO short_calls
      VALUES (caller_id, call_timestamp, call_duration)
  WHEN call_duration >= 50 AND call_type = 'tlk' THEN
    INTO long_calls
      VALUES (caller_id, call_timestamp, call_duration)
SELECT caller_id, call_timestamp, call_duration, call_format,
       recipient_caller
FROM calls
WHERE TRUNC(call_timestamp) = TRUNC(SYSDATE);

```

## Section 13 – DDL

### 13-1 Creating Tables

<b>Data dictionary</b>	Created and maintained by the Oracle Server and contains information about the database
<b>Schema</b>	A collection of objects that are the logical structures that directly refer to the data in the database
<b>DEFAULT</b>	Specifies a preset value if a value is omitted in the INSERT statement
<b>Table</b>	Stores data; basic unit of storage composed of rows and columns
<b>CREATE TABLE</b>	Command used to make a new table

Table names are not case sensitive.

Table names should be plural, for example STUDENTS, not student

The main database object types are:

Table	Index	Constraint	View	Sequence	Synonym
-------	-------	------------	------	----------	---------

```
CREATE TABLE my_cd_collection (  
  cd_number NUMBER(3),  
  title VARCHAR2(20) not null,  
  artist VARCHAR2(20) check(regex_like(artist, '[a-zA-Z .]')),  
  purchase DATE DEFAULT SYSDATE);
```

#### -- External Tables Example

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

User tables:	Data Dictionary tables (Only Select):
Employees Departments	SELECT * FROM DICTIONARY; SELECT * FROM USER_TABLES; SELECT * FROM USER_INDEXES; SELECT * FROM user_objects WHERE object_type= 'SEQUENCE'; SELECT * FROM USER_SEGMENTS; SELECT * FROM ALL_TABLES;

## 13-2 Using Data Types

<b>BLOB</b>	Binary large object data up to 4 gigabytes
<b>CLOB</b>	Character data up to 4 gigabytes
<b>INTERVAL YEAR TO MONTH</b>	Allows time to be stored as an interval of years and months
<b>INTERVAL DAY TO SECOND</b>	Allows time to be stored as an interval of days to hours, minutes, and seconds
<b>TIMESTAMP</b>	Allows the time to be stored as a date with fractional seconds
<b>TIMESTAMP WITH TIMEZONE</b>	stores a time zone value as a displacement from Universal Coordinated Time or UCT
<b>TIMESTAMP WITH LOCAL TIMEZONE</b>	when a column is selected in a SQL statement the time is automatically converted to the user's timezone

- CHAR (fixed size, maximum 2000 characters)
- VARCHAR2 (variable size, maximum 4000 characters)
- NUMBER (variable size, maximum precision 38 digits)
- DATE range yyyy-mm-dd hh24:mi:ss
- TIMESTAMP range yyyy-mm-dd hh12:mi:ss and fractions of a second
- INTERVAL DAY [(day\_precision)] TO SECOND The default precision value is 2

**select current\_timestamp, SYSTIMESTAMP from dual**

current_timestamp	03-OCT-22 05.22.33.598000000 PM AMERICA/MEXICO_CITY
SYSTIMESTAMP UCT	03-OCT-22 05.22.33.598000000 PM -05:00

MySQL Date yyyy-mm-dd	ORACLE Date yyyy-mm-dd hh:mi:ss
create table tmp_Formatos( Fecha date, FechaTiempo datetime, TiempoMarca timestamp);	create table tmp_Formatos( Fecha date, TiempoMarca timestamp);
select now(), sysdate(), current_timestamp();	select sysdate, current_date, current_timestamp, SYSTIMESTAMP from dual;
insert into tmp_Formatos values(sysdate(), sysdate(), sysdate());	insert into tmp_Formatos values(sysdate, sysdate);
Select * from tmp_formatos;	select * from tmp_formatos;
select second(fechaTiempo), extract(second from TiempoMarca) from tmp_formatos;	select to_char(fecha, 'ss'), extract(second from TiempoMarca) from tmp_formatos;

create table tmp_Horarios ( Fecha date, TS TIMESTAMP, TS_TZ TIMESTAMP WITH TIME ZONE, TS_LTZ TIMESTAMP WITH LOCAL TIME ZONE);	create table tmp_Intervalos ( loan1 INTERVAL YEAR TO MONTH, loan2 INTERVAL YEAR TO MONTH);
insert into tmp_horarios values (sysdate, sysdate, SYSTIMESTAMP, sysdate);	INSERT INTO tmp_Intervalos (loan1, loan2) VALUES (INTERVAL '121' MONTH(3), INTERVAL '3-6' YEAR TO MONTH);
	select sysdate+loan1 from tmp_intervalos;

## 13-3 Modifying a Table

You can add or modify a column in a table, but you cannot specify where the column appears

```
ALTER TABLE tablename
ADD (column_name data_type [DEFAULT expression],
     column_name data_type [DEFAULT expression], ...);
```

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

```
ALTER TABLE tablename
DROP COLUMN columnname;
```

```
-- Dropping a column from a large table can take a long time
ALTER TABLE tablename SET UNUSED (column_name);
```

```
-- when you want to reclaim the extra disk space
ALTER TABLE copy_employees
DROP UNUSED COLUMNS;
```

```
ALTER SESSION SET RECYCLEBIN = ON;
DROP TABLE table_name;
```

```
-- Recovery a Table
FLASHBACK TABLE table_name TO BEFORE DROP;
```

```
-- Show deleted tables
select * from USER_RECYCLEBIN;
```

```
-- Drop a table definitely
DROP TABLE Table_Name PURGE;
```

```
-- Rename a table
RENAME old_name to new_name;
```

it does not release storage space	Free up storage space
Delete from Table_Name;	Truncate Table Table_Name;

```
COMMENT ON TABLE Employees is 'Tabla de empleados';
comment on column Employees.last_name is 'Apellido Paterno';
```

```
select * from user_tab_comments;
SELECT * FROM USER_COL_COMMENTS;
```

	Review the changes made (UNDO tablespace) SCN (System Change Number)
UPDATE EMPLOYEES SET LAST_NAME = 'King Kong' where employee_id = 100;	select * from Employees VERSIONS BETWEEN SCN MINVALUE AND MAXVALUE WHERE employee_id= 100;



## Section 14 – Constraints

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

Constraint	Database rule.
<b>PRIMARY KEY</b>	Constraint ensures that the column contains no null values and uniquely identifies each row of the table
<b>UNIQUE KEY</b>	An integrity constraint that requires every value in a column or set of columns be unique
<b>UNIQUE constraint</b>	Every value in a column or set of columns (a composite key) must be unique
<b>FOREIGN KEY</b>	Designates a column (child table) that establishes a relationship between a primary key in the same table and a different table (parent table)
<b>REFERENCES</b>	Identifies that table and column in the parent table
<b>NOT NULL constraint</b>	For every row entered into the table, there must be a value for that column
<b>CHECK constraint</b>	Specifies a condition for a column that must be true for each row of data
<b>Table level constraint</b>	References one or more columns and is defined separately from the definitions of the columns in the table
<b>Column-level constraint</b>	Database rule that references a single column

5 Types of constraints: All the constraints have a name

NOT NULL	PRIMARY KEY	FOREIGN KEY	UNIQUE	CHECK
----------	-------------	-------------	--------	-------

There are two different places in the CREATE TABLE statement that you can specify the constraint details:

- At the **column level** next to the name and data type
- At the **table level** after all the column names are listed

Constraints at the Column Level	Constraints at the Table Level
<pre>CREATE TABLE clients (   IDnumber NUMBER(4) primary KEY,   LastName VARCHAR2(20) constraint nn_LN not null,   Email VARCHAR2(20) UNIQUE,   HireDate date default sysdate,   Salary number(6,2) check(salary &gt; 0) );</pre> <p>system gives the constraint a name, such as <b>SYS_C00585417</b></p> <p>The NOT NULL constraint can be specified only at the column level, not the table level</p>	<pre>CREATE TABLE clients (   IDNumber NUMBER(4),   LastName VARCHAR2(20),   Email VARCHAR2(20),   HireDate date default sysdate,   Salary number(6,2),   CONSTRAINT <b>Clients_IDNumber_pk</b> primary key (IDNumber),   CONSTRAINT <b>uk_Email</b> UNIQUE(Email),--unique(email,phone)   CONSTRAINT <b>check_Salary</b> check(Salary&gt;0) );</pre> <pre>ALTER TABLE clients MODIFY (LastName NOT NULL);</pre>

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

<b>PRIMARY KEY constraint</b>	A column or set of columns that uniquely identifies each row in a table
<b>FOREIGN KEY constraint</b>	Establishes a relationship between the foreign key column and a primary key or unique key in the same table or a different table
<b>NOT NULL</b>	Constraint ensures that the column contains no null values
<b>CHECK constraint</b>	Explicitly defines a condition that must be met
<b>ON DELETE SET NULL</b>	Allows a child row to remain in a table with null values when a parent record has been deleted
<b>ON DELETE CASCADE</b>	Allows a foreign key row that is reference to a primary key row to be deleted

### Column-level syntax example:

```
CREATE TABLE employees(
employee_id NUMBER(6,0) CONSTRAINT emp_pk PRIMARY KEY,
first_name VARCHAR2(20),
last_name VARCHAR2(25),
department_id NUMBER(4,0) CONSTRAINT emp_department_id_fk
REFERENCES departments(department_id) ON DELETE SET NULL,
email VARCHAR2(25));
```

### Table-level syntax example:

```
CREATE TABLE employees(
employee_id NUMBER(6,0) CONSTRAINT emp_pk PRIMARY KEY,
first_name VARCHAR2(20),
last_name VARCHAR2(25),
department_id NUMBER(4,0),
email VARCHAR2(25),
CONSTRAINT emp_dept_id_fk FOREIGN KEY (department_id)
REFERENCES departments(department_id) ON DELETE CASCADE);
```

### Composite primary key

<pre>CREATE TABLE job_history ( employee_id NUMBER(6,0), start_date DATE, end_date DATE, job_id VARCHAR2(10), department_id NUMBER(4,0), CONSTRAINT jh_pk PRIMARY KEY(employee_id, start_date), CONSTRAINT jh_endDate_ck CHECK (end_date &gt; start_date));</pre>	<p>CHECK constraint cannot contain functions: SYSDATE, UID, USER, or USERENV Example: SYSDATE &gt; '05-May-1999'</p> <p>CHECK constraint cannot use: CURRVAL, NEXTVAL, LEVEL, or ROWNUM</p>
---	---

## 14-3 Managing Constraints

<b>DISABLE CONSTRAINT</b>	To deactivate an integrity constraint
<b>CASCADE clause</b>	Disables dependent integrity constraints
<b>ALTER TABLE</b>	To add, modify, or drop columns from a table
<b>ENABLE CONSTRAINT</b>	To activate an integrity constraint currently disabled
<b>DROP CONSTRAINT</b>	Removes a constraint from a table
<b>DROP COLUMN</b>	Allows user to delete a column from a table
<b>CASCADE CONSTRAINT clause</b>	Defines the actions the database server takes when a user attempts to delete or update a key to which existing foreign keys point

Write ALTER TABLE statements to add, drop, disable, and enable constraints

<pre>select * from <b>USER_CONSTRAINTS</b> WHERE TABLE_NAME='EMPLOYEES';  select * from USER_CONS_COLUMNS where table_name = 'EMPLOYEES';</pre>	<pre>* constraint_name, table_name, constraint_type, status <b>constraint_type:</b> P PRIMARY KEY R REFERENCES (foreign key); C CHECK constraint (including NOT NULL); U UNIQUE</pre>
---	---

Sintaxis	Example
<pre>ALTER TABLE table_name ADD [CONSTRAINT constraint_name]     type of constraint (column_name);</pre>	<pre>ALTER TABLE employees ADD CONSTRAINT emp_id_pk     PRIMARY KEY (employee_id);</pre>
<pre>ALTER TABLE tablename ADD [CONSTRAINT constraint_name]     FOREIGN KEY(column_name) REFERENCES tablename(column_name);</pre>	<pre>ALTER TABLE employees ADD CONSTRAINT emp_dept_fk     FOREIGN KEY (department_id) REFERENCES departments (department_id);</pre>
<pre>ALTER TABLE table_name <b>MODIFY</b> (column_name CONSTRAINT     constraint_name <b>NOT NULL</b>);</pre>	<pre>ALTER TABLE employees MODIFY (email CONSTRAINT     emp_email_nn NOT NULL);</pre>
<pre>ALTER TABLE table_name DROP CONSTRAINT name [CASCADE]</pre>	<pre>ALTER TABLE departments DROP CONSTRAINT dept_dept_id_pk;</pre>

Sintaxis	Example
<pre>ALTER TABLE table_name <b>DISABLE CONSTRAINT</b> constraint_name [CASCADE]</pre>	<pre>ALTER TABLE departments DISABLE CONSTRAINT dept_dept_id_pk;</pre>
<pre>ALTER TABLE table_name <b>ENABLE CONSTRAINT</b> constraint_name;</pre>	<pre>ALTER TABLE departments ENABLE CONSTRAINT dept_dept_id_pk;</pre>

## Section 15 – Views

### 15-1 Creating Views

View	A subset of data from one or more tables that is generated from a query and stored as a virtual table
VIEW_NAME	Name of view
CREATE VIEW	Statement used to create a new view
REPLACE	Re-creates the view if it already exists
NOFORCE (default)	Creates the view only if the base table exists
FORCE	Creates a view regardless of whether or not the base tables exist
Alias	Specifies a name for each expression selected by the view's query
Subquery	A complete SELECT statement
Simple view	Derives data from one table, no functions or groups, performs DML operations through the view
Complex view	Derives data from one or more tables, contains functions or groups of data, and does not always allow DML operations through the view
CONSTRAINT	Is the name assigned to the CHECK OPTION constraint.
WITH READ ONLY	Ensures that no DML operations can be performed on this view.
WITH CHECK OPTION	Specifies that rows remain accessible to the view after insert or update operations.

Views restrict access to base table data because the view can display selective columns from the table

Views can be used to reduce the complexity of executing queries based on more complicated SELECT statements

<pre>CREATE [OR REPLACE] [FORCE  NOFORCE] VIEW view [(alias [,alias]...)] AS subquery [WITH CHECK OPTION [CONSTRAINT constraint]] [WITH READ ONLY [CONSTRAINT constraint]];</pre>
---

Feature	Simple Views	Complex Views
Number of tables used to derive data	One	One or more
Can contain functions	No	Yes
Can contain groups of data	No	Yes
Can perform DML operations (INSERT, UPDATE, DELETE) through a view	Yes	Not always

Simple View	
<pre>CREATE OR REPLACE VIEW view_euro_countries AS SELECT country_id ID, country_name "Country" FROM wf_countries WHERE location LIKE '%Europe';</pre>	<pre>CREATE OR REPLACE VIEW view_euro_countries(ID, "Country") AS SELECT country_id, country_name FROM wf_countries WHERE location LIKE '%Europe';</pre>

Complex View	
<pre>CREATE OR REPLACE VIEW view_euro_countries ("ID", "Country", "Region") AS SELECT c.country_id, c.country_name, r.region_name FROM wf_countries c JOIN wf_world_regions r USING (region_id) WHERE location LIKE '%Europe';</pre>	<pre>CREATE OR REPLACE VIEW view_high_pop ("Region ID", "Highest population") AS SELECT region_id, MAX(population) FROM wf_countries GROUP BY region_id;</pre>

SELECT \* FROM view\_high\_pop;

## 15-2 DML Operations and Views

<b>ROWNUM</b>	A pseudo-column which assigns a sequential value starting with 1 to each of the rows returned from the subquery
<b>WITH READ ONLY</b>	Ensures that no DML operations can be performed on this view
<b>WITH CHECK OPTION</b>	Specifies that INSERTS and UPDATES performed through the view can't create rows which the view cannot select

<pre>CREATE OR REPLACE VIEW view_dept50 AS SELECT department_id, employee_id, salary FROM employees WHERE department_id = 50 <b>WITH CHECK OPTION</b> CONSTRAINT view_dept50_check;</pre>	<pre>UPDATE view_dept50 SET salary = 5800, <b>BONUS = 0.5</b> WHERE employee_id= 124;</pre>
---	---

<pre>CREATE OR REPLACE VIEW view_dept50 AS SELECT department_id, employee_id, salary FROM employees WHERE department_id = 50 <b>WITH READ ONLY;</b></pre>	<pre>UPDATE view_dept50 SET salary = 5800 WHERE employee_id= 124;</pre>
---	---

## 15-3 Managing Views

<b>DROP VIEW</b>	Removes a view
<b>INLINE VIEW</b>	Subquery with an alias that can be used within a SQL statement
<b>TOP- N-ANALYSIS</b>	Asks for the N largest or smallest values in a column

Only the creator or users with the DROP ANY VIEW privilege can remove a view:

**DROP VIEW viewname**

INLINE VIEW	TOP_N, LIMIT
<pre>SELECT e.department_id, e.employee_id, d.minsal FROM employees e,       (SELECT department_id, min(salary) minsal        FROM employees        GROUP BY department_id) d WHERE e.department_id = d.department_id AND e.salary= d.minsal;</pre>	<pre>SELECT ROWNUM AS "Top_N", employee_id, last_name FROM (SELECT employee_id, last_name       FROM employees       ORDER BY last_name) WHERE ROWNUM &lt;=5;</pre>

<pre> with   min_dep as (SELECT department_id, min(salary) minsal               FROM employees               GROUP BY department_id) select e.department_id, e.employee_id, d.minsal from employees e, min_dep d where e.department_id = d.department_id and e.salary= d.minsal; </pre>	
---	--

Mostrar los empleados, con los departamentos que tienen más de un empleado, con salario mínimo por depto.

<pre> with dmin as (SELECT department_id, min(salary) minsal               FROM employees               GROUP BY department_id),   emin as (SELECT e.department_id, e.employee_id, dmin.minsal            FROM employees e, dmin            WHERE e.department_id = dmin.department_id                AND e.salary= dmin.minsal),   dmas as (select department_id, count(*)           from emin           group by department_id           having count(*) &gt; 1 ) select emin.* from emin, dmas where emin.department_id = dmas.department_id; </pre>
---

## Section 16 – Sequences and Synonyms

### 16-1 Working With Sequences

<b>CREATE SEQUENCE</b>	Command that automatically generates sequential number
<b>Sequences</b>	Generates a numeric value
<b>STARTS WITH</b>	Specifies the first sequence number to be generated
<b>INCREMENT BY</b>	Specifies the interval between sequence numbers
<b>CURRVAL</b>	Returns the current sequence value
<b>NEXTVAL</b>	Returns the next available sequence value
<b>MINVALUE</b>	Specifies the minimum sequence value
<b>MAXVALUE</b>	Specifies a maximum or default value the sequence can generate
<b>CYCLE/ NOCYCLE</b>	Specifies whether the sequence continues to generate values after reaching its maximum or minimum values
<b>NOMAXVALUE</b>	Specifies a maximum value of $10^{27}$ for an ascending sequence and -1 for a descending sequence (default)
<b>NOMINVALUE</b>	Specifies a minimum value of 1 for an ascending sequence and $-(10^{26})$ for a descending sequence (default)
<b>CACHE/ NOCACHE</b>	Specifies how many values the Server pre-allocates and keeps in memory. (By default, the Oracle server caches 20 values.)

SELECT sequence\_name, min\_value, max\_value, increment\_by, last\_number

FROM **user\_sequences**;

CREATE SEQUENCE sequence [INCREMENT BY n] [START WITH n] [{MAXVALUE n   NOMAXVALUE}] [{MINVALUE n   NOMINVALUE}] [{CYCLE   NOCYCLE}] [{CACHE n   NOCACHE}];	CREATE SEQUENCE runner_id_seq INCREMENT BY 1 START WITH 1 MAXVALUE 50000 NOCACHE NOCYCLE;  ALTER SEQUENCE runner_id_seq INCREMENT BY 1 MAXVALUE 999999;  DROP SEQUENCE runner_id_seq;	select runner_id_seq.CURRVAL from dual;  -- Quemar Folio select runner_id_seq.NEXTVAL from dual;  SELECT last_number as Next FROM USER_SEQUENCES WHERE sequence_name= 'RUNNER_ID_SEQ';
---	--	---

INSERT INTO departments (department_id, department_name, location_id) VALUES (departments_seq.NEXTVAL, 'Support', 2500);
--

## 16-2 Indexes and Synonyms

<b>Synonym</b>	Gives alternative names to objects
<b>CREATE PUBLIC SYNONYM</b>	To refer to a table by another name to simplify access
<b>Composite index</b>	An index that you create on multiple columns in a table
<b>Unique index</b>	The Oracle Server automatically creates this index when you define a column in a table to have a PRIMARY KEY or a UNIQUE KEY constraint
<b>Non-unique index</b>	Schema object that speeds up retrieval of rows
<b>DROP INDEX</b>	Removes an index
<b>Confirming index</b>	Confirms the existence of indexes from the USER_INDEXES data dictionary view
<b>Function-based index</b>	Stores the indexed values and uses the index based on a SELECT statement to retrieve the data

index	Synonym
Select * from USER_INDEXES;	Select * from USER_SYNONYMS;
Select * from USER_IND_COLUMNS;	
CREATE [unique] INDEX index_name_idx ON table_name (column ..., column)  ■ Function-based Indexes CREATE INDEX upper_last_name_idx ON employees (UPPER(last_name));	CREATE [PUBLIC] SYNONYM synonym FOR object;  CREATE SYNONYM amy_emps FOR amy_copy_employees;
DROP INDEX upper_last_name_idx;	DROP [PUBLIC] SYNONYM name_of_synonym; DROP SYNONYM amy_emps;



## Section 17 – Privileges and Regular Expressions

### 17-1 Controlling User Access

Grant and Revoke privileges.

Compare the difference between **object privileges** and **system privileges**.

Database security can be classified into two categories: **Data security** and **System security**.

<b>Data security</b> (also known as object security) relates to object privileges which covers access to and use of the database objects, and the actions that those users can have on the objects. These privileges include being able to execute DML statements	<b>System security</b> covers access and use of the database at the system level, such as creating users, allocating disk space to users, and granting the system privileges that users can perform such as creating tables, views, and sequences.
---	--

Roles: are named groups of related privileges.

The **DBA**: is a high-level user with the ability to grant users access to the database and its objects.

A **schema** is a collection of objects, such as tables, views, and sequences. The schema is owned by a database user and has the same name as that user.

System Privilege	Operations Authorized
CREATE USER	Grantee can create other Oracle users (a privilege required for a DBA role)
DROP USER	Grantee can drop another user
DROP ANY TABLE	Grantee can drop a table in any schema
BACKUP ANY TABLE	Grantee can backup any table in any schema with the export utility
SELECT ANY TABLE	Grantee can query tables, views, or snapshots in any schema
CREATE ANY TABLE	Grantee can create tables in any schema

CREATE USER user IDENTIFIED BY password;	ALTER USER user IDENTIFIED BY password;
---	--

#### Granting System Privileges

<pre>GRANT privilege [, privilege...] TO user [, user  role, PUBLIC...];  GRANT create session, create table, create sequence, create view TO Scott; GRANT UPDATE (salary) ON employees TO Scott; GRANT select ON scott.departments TO PUBLIC;</pre>
--

You can't grant SELECT on individual columns, but you can create a VIEW and granting SELECT privilege

System Privilege	Operations Authorized
CREATE SESSION	Connect to the database
CREATE TABLE	Create tables in the user's schema
CREATE SEQUENCE	Create a sequence in the user's schema
CREATE VIEW	Create a view in the user's schema
CREATE PROCEDURE	Create a procedure, function, or package in the user's schema

Object Privilege	Table	View	Sequence	Procedure
ALTER	X		X	
DELETE	X	X		
EXECUTE				X
INDEX	X	X		
INSERT	X	X		
REFERENCES	X			
SELECT	X	X	X	
UPDATE	X	X		

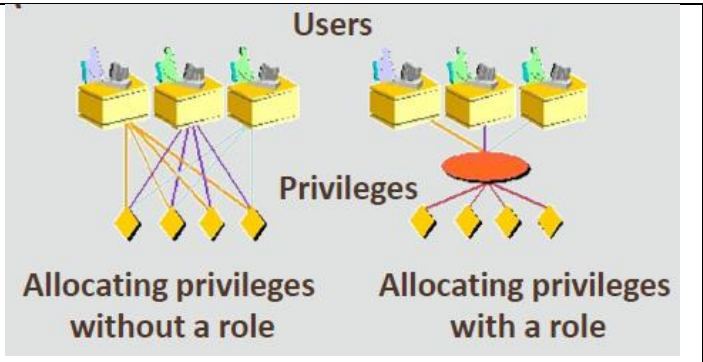
Data Dictionary View	Description
ROLE_SYS_PRIVS	System privileges granted to roles
ROLE_TAB_PRIVS	Table privileges granted to roles
USER_ROLE_PRIVS	Roles accessible by the user
USER_TAB_PRIVS_MADE	Object privileges granted on the user's objects
USER_TAB_PRIVS_RECD	Object privileges granted to the user
USER_COL_PRIVS_MADE	Object privileges granted on the columns of the user's objects
USER_COL_PRIVS_RECD	Object privileges granted to the user on specific columns
USER_SYS_PRIVS	Lists system privileges granted to the user

### 17-2 Creating and Revoking Object Privileges

A role is a named group of related privileges that can be granted to a user.

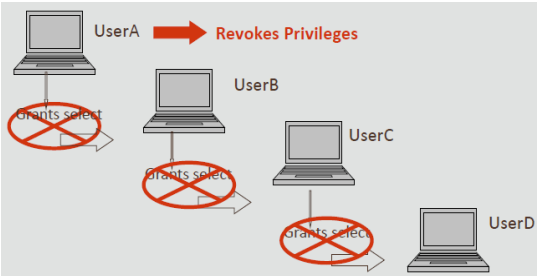
- To create and assign a role, first the DBA must create the role
- Then the DBA can assign privileges to the role, and the role to users.

```
CREATE ROLE manager;
GRANT create table, create view TO manager;
GRANT manager TO Scott;
```



<p>Granting Object Privileges</p> <p>Construct a GRANT..ON..TO..WITH GRANT OPTION</p> <pre>GRANT object_priv[(column_list)] ON object_name TO {user role PUBLIC} [WITH GRANT OPTION];</pre>	<p>Scott is the owner:</p> <pre>GRANT SELECT, insert ON clients TO PUBLIC;  GRANT UPDATE(first_name, last_name) ON clients TO King, manager;</pre> <p>King execute:</p> <pre>Select * from Scott.clients;</pre>
---	---

Syntax	Defined
object_priv	is an object privilege to be granted
column_list	specifies a column from a table or view on which privileges are granted
ON object_name	is the object on which the privileges are granted
TO user role	identifies the user or role to whom the privilege is granted
PUBLIC	grants object privileges to all users
WITH GRANT OPTION	Allows the grantee to grant the object privileges to other users and roles. Permite al beneficiario otorgar privilegios de objeto a otros usuarios y roles.  If the owner revokes a privilege from a user who granted privileges to other users, the revoke statement cascades to all privileges granted

<p>■ Revoking Object Privileges</p> <pre> REVOKE {privilege [, privilege...] ALL} ON object FROM {user[, user...] role PUBLIC} [CASCADE CONSTRAINTS]; </pre>	<p>if user A revokes privileges from user B, then those privileges granted to users C and D are also revoked.</p> 
--	--

#### Database Links.

A local user can access a remote database without having to be a user on the remote database

<pre> Select * form USER_DB_LINKS;  CREATE PUBLIC SYNONYM HQ_EMP FOR emp@HQ.ACME.COM;  SELECT * FROM HQ_EMP; </pre>	
---	--

### 17-3 Regular Expressions

LIKE and wildcards \_ %

The use of regular expressions is based on the use of meta characters

Symbol	Description
.	(dot) Matches any character in the supported character set, except NULL
?	Matches zero or one occurrence
*	Matches zero or more occurrences
+	Matches one or more occurrences
()	Grouping expression, treated as a single sub-expression
\	Escape character
	Alternation operator for specifying alternative matches
^/\$	Matches the start-of-line/end-of-line
[ ]	Bracket expression for a matching list matching any one of the expressions represented in the list

Where column like 'a\_c' = Where REGEXP\_LIKE(column, 'a.c')

Name	Description
REGEXP_LIKE	Similar to the LIKE operator, but performs regular expression matching instead of simple pattern matching
REGEXP_REPLACE	Searches for a regular expression pattern and replaces it with a replacement string
REGEXP_INSTR	Searches for a given string for a regular expression pattern and returns the position where the match is found
REGEXP_SUBSTR	Searches for a regular expression pattern within a given string and returns the matched substring
REGEXP_COUNT	Returns the number of times a pattern appears in a string. You specify the string and the pattern. You can also specify the start position and matching options (for example, c for case sensitivity).

```
SELECT first_name, last_name
FROM employees
WHERE REGEXP_LIKE(first_name, '^Ste(v|ph)en$');

CREATE TABLE my_contacts(
first_name VARCHAR2(15),
last_name VARCHAR2(15),
email VARCHAR2(30) CHECK(REGEXP_LIKE(email, '.*@.*\..*'))
);
```

## Section 18 – TCL

### 18-1 Database Transactions