Create View with select

Format of the state:
CREATE  [OR  REPLACE]  VIEW  #view_name#  
[([#newColName#, ...])]
AS  defining_query  [WITH  [[CASCADED|LOCAL]
CHECK  OPTION]|READ  ONLY]

Number of columns listed must match number of columns return in the query

CREATE  object  [IF  NOT  EXISTS]
DROP  object  [IF  EXISTS]
VIEW

It’s possible to add, update, remove from view in the base, if
- Consists of a single table, no join, union, intersection, or except (difference)
- Distinct not specified
- Every element in the SELECT is a column name (not constant nor derived), appear only once
- No corelated subquery
- No group by or having clause
- Columns excluded either allow NULL value or has default value

Any updates to a base table are immediately reflected in the view, and vice versa.
VIEW

CHECK OPTION with Where clause in the defining query

Prevents update/insert rows that exist in a view disappear from the view.

LOCAL vs CASCADED (default)

CREATE VIEW HONORSTUDENT
AS SELECT *
FROM STUDENT
WHERE GPA >= 3.5
WITH CHECK OPTION;

Not valid:
UPDATE HONORSTUDENT
SET GPA = 3.2
WHERE CUNY_ID = ‘12345678’;
CREATE VIEW CUMLAUDE
AS SELECT *
    FROM STUDENT
    WHERE GPA >= 3.5;

CREATE VIEW MAGNA
AS SELECT *
    FROM CUMLAUDE
    WHERE GPA >= 3.75
WITH LOCAL CHECK OPTION;

CREATE VIEW SUMMA
AS SELECT *
    FROM MAGNA
    WHERE GPA >= 3.85;

Not valid:
UPDATE SUMMA
SET GPA = 3.4 ← VALID
WHERE CUNY_ID = ‘12345678’;
CREATE VIEW CUMLAUDE
AS SELECT *
FROM STUDENT
WHERE GPA >= 3.5;

CREATE VIEW MAGNA
AS SELECT *
FROM CUMLAUDE
WHERE GPA >= 3.75
WITH LOCAL CHECK OPTION;

CREATE VIEW SUMMA
AS SELECT *
FROM MAGNA
WHERE GPA >= 3.85;

Not valid:
UPDATE SUMMA
SET GPA = 3.4 ← VALID
WHERE CUNY_ID = ‘12345678’;
PL/SQL

Extension SQL - PROGRAMMABLE LANGUAGE
DDL and DML are not computationally complete
SQL/PSM (Persistent Stored Modules)

Oracle programmable procedural language/SQL
PL/SQL

It’s block-structured language:
- Declaration (optional) – variables, constants, cursors, exceptions
BEGIN
- Executable part (mandatory)
- Exception handling (optional)
END;
Declarations

variable_name [CONSTANT] type [NOT NULL]

type is column data type, may copy the type from another variable_type/column_type use #var_or_col#%TYPE

Variable may store entire row and get the type from %ROWTYPE
Note: %type and %rowtype are not standard SQL. NOT NULL must has an initial value

Sample,
vStuID CHAR(8);
vGpa Student.gpa%TYPE;
vStuID2 vStuID%TYPE;

vStudent Student%ROWTYPE;
Assignment

Operator := or from the result of an SQL SELECT or FETCH statement

Sample,

vStuID CHAR(8) := '10000001';

vMaxGpa Student.gpa%TYPE NOT NULL:= 0;

vPi CONSTANT NUMBER(3,2) := 3.14;

vStuID := '87654321';

SELECT MAX(GPA) INTO vMaxGpa FROM STUDENT;
PL/SQL

Sample:

```plsql
SET SERVEROUTPUT ON
DECLARE
vSID STUDENT.cuny_id%type;
BEGIN
SELECT cuny_id into vSID FROM student
WHERE F_NAME LIKE 'J%';
dbms_output.put_line(vSID); /*DEBUG LINE*/
END;
```
Control Statement

Conditional

IF (condition) THEN
  statement
ELSIF (condition) THEN
  statement
ELSE
  statement
END IF;
Control Statement

Conditional
CASE (operand)
WHEN (condition|value) THEN
...list
ELSE
END CASE;

CASE vStuID
  WHEN '12345678' THEN vGpa := 3.7;
  WHEN '87654321' THEN vGpa := 3.8;
  vStuID := '10000002';
END CASE;

UPDATE STUDENT
SET GPA = CASE
  WHEN CUNY_ID = '12345678'
    THEN 3.7
  ELSE
    3.8
END:
Iterative Statement

iterative
 [NAME]
LOOP
  statement
EXIT [NAME] WHEN(condition)
END LOOP [NAME];

WHILE (condition) LOOP
  statement
END LOOP [NAME];

FOR idxVariable
  IN l .. h LOOP
  statement
END LOOP [NAME];
Iterative Statement

Sample:
DECLARE
x INTEGER := 1;
BEGIN
LOOP
  x := x + 1;
  DBMS_OUTPUT.PUT_LINE(x);
  EXIT WHEN x = 10;
END LOOP;
END;
Iterative Statement

Sample:
DECLARE
  i number(1);
  j number(1);
BEGIN
  << outer_loop >>
  FOR i IN 1..3 LOOP
    << inner_loop >>
    FOR j IN 1..3 LOOP
      dbms_output.put_line('i: ' || i || ', j: ' || j);
    END LOOP
  END LOOP
END;

MULTIPLE ROWS

CURSOR (POINTER)
- WHEN A QUERY RETURNS 0 OR MORE ROWS.
Be declared and opened before use, close after done.

1. DECLARE CURSOR
   CURSOR #cname# IS

2. OPEN CURSOR
   OPEN #cname#

3. FETCH CURSOR
   FETCH #cname# INTO #vname#, ...

4. CLOSE
   CLOSE #cname
MULTIPLE ROWS

Sample,

DECLARE
    sgpa STUDENT.GPA%TYPE;
    sid STUDENT.CUNY_ID%TYPE;
CURSOR studentCursor IS
    SELECT CUNY_ID, GPA
    FROM STUDENT
    ORDER BY GPA DESC;
BEGIN
    OPEN studentCursor;
    LOOP
        FETCH studentCursor INTO sid, sgpa;
        EXIT WHEN studentCursor%NOTFOUND;
        dbms_output.put_line(sid || ': ' || sgpa);
    END LOOP;
    CLOSE studentCursor;
END;
Trigger

An action that executed automatically when an event occurs. Insert, Update, Delete. (Create, Alter, Drop)
Enforce constrains or audit changes.

CREATE TRIGGER #trig_name#
BEFORE | AFTER | INSTEAD OF
INSERT | DELETE | UPDATE [OF trigCol]
ON #table_name#
[REFERENCING OLD|NEW AS #name#]
[FOR EACH ROW|STATEMENT]
[WHEN condition]
...block of trigger action
## Trigger

<table>
<thead>
<tr>
<th></th>
<th>NEW</th>
<th>OLD</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSERT</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>DELETE</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>UPDATE</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>
Sample Prevent Professor from teaching more than 20 classes

CREATE TRIGGER ProfessorTeachTooMuchClasses
BEFORE INSERT ON Class
REFERENCING NEW AS nC
FOR EACH ROW
DECLARE
  vCount INTEGER;
BEGIN
  SELECT COUNT(*) INTO vCount
  FROM CLASS
  WHERE pro_id = :nC.pro_id;
  IF vCount = 20 THEN
    dbms_output.putline('Too Many CLASSES');
  END IF;
END;