1) **ALTER TABLE**

```
alter table    table_name
add
   (
   column1_name column1_datatype column1_constraint,
   column2_name column2_datatype column2_constraint,
   column3_name column3_datatype column3_constraint
   );
```

- **Sa se adauge tabelului *salarii* coloana *prima* de tip int.**

alter table salarii
add prima int;

- **Sa se stearga coloana de la *prima*.**

alter table salarii
drop column prima;

- **Sa se adauge tabelului salarii coloana prima de tip int cu constrangerea ca prima sa fie mai mica sau egala decat 100.**

alter table salarii
add prima int check(prima<=100);

- **Modificarea coloanelor**

```
ALTER TABLE nume_tabela
  MODIFY(nume_coloana [tip_date]
    [DEFAULT expresie] [constrangere])
```

Prin această cerere se pot modifica tipul de date al coloanei, valoarea implicită si se poate adăuga/modifica o constrângere de tip NULL / NOT NULL pentru acea coloană.

- **Sa se modifice coloana prima astfel incat sa fie de tipul number (7,2) (7 cifre dintre care doua dupa virgula).**

alter table salarii
modify (prima number(7,2));

- **Adaugare constrangeri**

```
ALTER TABLE nume_tabela
ADD [CONSTRAINT nume] tip(coloana);
```

Exemplu:

ALTER TABLE ANGAJATI
ADD CONSTRAINT NUME_3 CHECK(LENGTH(NUME)>3);

ALTER TABLE ANGAJATI
drop constraint Nume_3

- **ACTIVARE/DEZACTIVARE CONSTRANGERE**

Sintaxa:
Dezactivare:

```
ALTER TABLE nume_tabel DISABLE PRIMARY KEY [CASCADE]
ALTER TABLE nume_tabel DISABLE UNIQUE(lista_coloane)[CASCADE]
ALTER TABLE nume_tabel DISABLE CONSTRAINT [CASCADE]
```

Reactivare:

```
ALTER TABLE nume_tabel ENABLE PRIMARY KEY;
ALTER TABLE nume_tabel ENABLE UNIQUE(lista_coloane);
ALTER TABLE nume_tabel ENABLE CONSTRAINT nume;
```

2) Functii de prelucrare a valorii NULL

Functiile generale sunt: NVL, NVL2, NULLIF si COALESCE, aceste functii lucreaza cu orice tip de date.

Functie	Descriere
NVL (expr1, expr2)	Daca *expr1* nu este null, returneaza *expr1*. Daca *expr1* este null returneza *expr2*
NVL2 (expr1, expr2, expr3)	Daca *expr1* nu este null, NVL2 returneaza *expr2*. Daca *expr1* este null, NVL2 returneza *expr3*. Argumentul *expr1* poate fi orice tip de date.
NULLIF (expr1, expr2)	Compara doua expresii si returneaza valoarea null daca sunt egale sau prima expresie daca nu sunt egale.
COALESCE (expr1, expr2, ..., expr n)	Returneaza prima expresie non-null din lista de expresii.

Sa se afiseze din tabelul *salarii* coloanele *marca, salariu_baza, prima, total (=salariu_baza+prima)*.

select marca,salariu_baza,prima,salariu_baza+NVL(prima,0) as total
from salarii;
//sa adune cu 0 daca avem NULL

3) Exemple de utilizare a comenzii select

- Sa se utilizeze pseudocoloana *rownum* pt numerotarea randurilor rezultat al comenzii select.

Rownum este o pseudo-coloana. Aceasta pastreaza numarul inregistrarii, numerotarea inregistrarilor incepe de la 1.

select rownum as nrCrt, salarii.* from salarii order by marca desc;

- Sa se afiseze cele mai mari 10 salarii in ordinea descrescatoare a lor.

select rownum as nrCrt, salarii.* from salarii where rownum<=10 order by salariu_baza desc;

- Sa se determine totalul cheltuielilor salariale la nivel de departament.

```
select cod_dep, sum(salariu_baza+NVL(prima,0)) as TOTAL
        from angajati A inner join salarii B
        on A.marca=B.marca
        group by cod_dep;
```

- Sa se afiseze salariatii cu salariu_baza maxim.

```
select * from angajati A inner join salarii B on A.marca=B.marca
        where salariu_baza = ( select max(salariu_baza) from salarii
                                )
```

4) Comanda update

UPDATE table-Name
SET column-Name = Value
[, column-Name = Value,…]
[WHERE clause] |

- **Sa se acorde o prima de 100 lei salariatului cu marca=1;**

```
update salarii
        set prima=100
        where marca=1;
```

- Sa se acorde aceleasi drepturi salariale angajatului cu marca=2 ca si celui cu marca=1.

```
update salarii
set (salariu_baza, prima)=(select salariu_baza, prima
                            from salarii
                           where marca=1
                          )
where marca=2;
```

- Sa se acorde angajatului cu marca=2 salariul angajatului cu marca=1 cu 200lei mai putin si prima pe jumatate.

update salarii

set (salariu_baza, prima)=(select salariu_baza-200, prima/2
from salarii
where marca=1
)
where marca=2;

5) 12. Functia DECODE

- Aceasta este una dintre cele mai puternice functii Oracle actionand ca o comanda if-then-else sau case dintr-un limbaj procedural.

Forma generală:

DECODE(col | expresie, valoare1, rezultat1 [, valoare2, rezultat2 […]]
[, rezultat_implicit])

Dacă col | expresie ia valoarea $valoare_k$, atunci funcţia DECODE va returna valoarea $rezultat_k$.

Sintaxa funcţiei **CASE** este similară comenzii If-Then-Else. Oracle verifică fiecare condiţie începând cu prima condiţie (de la stânga la dreapta). Dacă condiţia respectivă este adevărată (partea WHEN), este returnată valoarea expresiei (partea THEN). Dacă nici o condiţie nu este verificată, se returnează expresia de la ELSE. Partea ELSE este opţională – expresia CASE va returna Null dacă nu este adevărată nici o condiţie.

```
CASE [expression]
     WHEN condition_1 THEN result_1
     WHEN condition_2 THEN result_2
     …
     WHEN condition_n THEN result_n
     ELSE result
END
```

- **Sa se mareasca salariul angajatului cu marca=1 cu 10%, agajatului cu marca=2 cu 15% si a celorlalti cu 5%.**

update salarii
set salariu_baza=salariu_baza* decode(marca, 1, 1.1, 2, 1.15, 1.05)
from salarii;

//decode este similar cu case din limbajele procedurale
//….???

```
select sal, case
                when sal < 2000 then 'category 1'
               when sal < 3000 then 'category 2'
               when sal < 4000 then 'category 3'
               else 'category 4'
          end
from emp;
```

- Sa se modifice *salariu_baza* pentru salariatii din departamentul *aprov* conform formulei: *salariu_baza=salariu_baza*1.1* ;

update salarii S
set salariu_baza=salariu_baza*1.1
where (select cod_dep from angajati A inner join salarii B on A.marca=B.marca
where A.marca=S.marca)='aprov';

6) CREAREA VEDERILOR

O sintaxă simplificată a comenzii `CREATE VIEW` este următoarea:

```
CREATE [OR REPLACE] [FORCE | NOFORCE] VIEW nume_vedere
[(alias [,alias]...)]
AS subinterogare
[WITH READ ONLY]
[WITH CHECK OPTION [CONSTRAINT nume_constrangere]]
```

unde

- `OR REPLACE` recreează vederea dacă ea există deja. Această opţiune poate fi folosită pentru a schimba definiţia unei vederi existente fără a o distruge în prealabil. Avantajul recreării vederii prin opţiunea `REPLACE` este că în acest caz se păstrează toate privilegiile acordate asupra acestei vederi. De exemplu, să presupunem că după crearea unei vederi, au fost acordate privilegii asupra vederii pentru anumite roluri sau pentru anumiţi utilizatori. Dacă aceea vederea este distrusă şi recreată, atunci toate privilegiile asupra vederii au fost pierdute şi trebuie acordate din nou. Dacă vederea este însă recreată folosind opţiunea `OR REPLACE`, atunci privilegiile acordate sunt păstrate şi nu mai este necesară acordarea lor încă o dată.
- `FORCE` este o opţiune care permite crearea vederii indiferent dacă tabelele de bază şi coloanele la care se face referire există sau nu, sau dacă utilizatorul posedă sau nu privilegiile corespunzătoare în legătură cu tabelele respective. Opţiunea opusă, `NOFORCE`, creează vederea numai dacă tabelele de bază există şi dacă utilizatorul posedă privilegiile corespunzătoare în legătură cu tabelele respective; `NOFORCE` este opţiunea implicită. Dacă se foloseşte opţiunea `FORCE` şi un tabel de bază nu există sau una dintre coloane nu este validă, atunci Oracle va crea vederea cu erori de compilare. Dacă mai târziu tabelul în cauză este creat sau coloana este corectată, atunci vederea poate fi folosită, Oracle recompilând-o dinamic înainte de folosire.
- `alias` specifică numele expresiilor selectate de interogarea vederii. Numărul alias-urilor trebuie să fie acelaşi cu numărul de expresii selectate de către interogarea vederii. Un alias trebuie să fie unic în cadrul unei interogări. Dacă sunt omise alias-urile, Oracle va folosi denumirile coloanelor din interogare. Atunci când interogarea vederii conţine şi expresii, nu doar simple coloane, trebuie folosite alias-uri.
- `AS` indică interogarea vederii. Aceasta poate fi orice instrucţiune `SELECT` care nu conţine clauzele `ORDER BY` şi `FOR UPDATE`.
- opţiunea `WITH READ ONLY` asigură că nici o operaţie DML (inserare, ştergere, modificare) nu va fi asigurată asupra vizualizării.

- `WITH CHECK OPTION` este o constrângere care arată că toate actualizările efectuate prin intermediul vederii vor afecta tabelele de bază numai dacă actualizările respective vor avea ca rezultat numai rânduri care pot fi vizualizate prin intermediul vederii. `CONSTRAINT` furnizează un nume pentru constrângerea `CHECK OPTION`. Asupra acestor opţiuni vom reveni puţin mai târziu

Distrugerea vederilor

Pentru distrugerea unei vederi se foloseşte comanda `drop view`:

```
drop view nume_vedere;
```

- Sa se creeze vederea *v_Angajati* pe baza datelor din tabelul angajati din schema RU si salarii si schema SAL. Vederea se va crea in schema SAL.

create view v_Angajati as
select A.*, salariu_baza, data_angajarii, vechime_anterioara
from angajati A inner join salarii B on A.marca=B.marca;
(am folosit synonym-ul *angajati* ***al tabelului*** *ru.angajati* ***)***
select * from v_Angajati;

- Sa se creeze tabelul *t_Aprovizionare* pe baza datelor din vederea *v_Angajati*.

create table t_Aprovizionare as
select* from v_Angajati;

- Sa se stearga tabelul de la 2.

drop table t_Aprovizionare;

- Sa se creeze tabelul *t_Ang_Aprovizionare* pe baza datelor din vederea *v_Angajati* ce contine salariatii din departamentul Aprov.

create table t_Ang_Aprovizionare as
select* from v_Angajati where cod_dep='aprov';

select * from t_Aprovizionare;

- Sa se afiseze salariatii cu salariu_baza maxim din fiecare departament.

select * from v_Angajati A
where salariu_baza=(select max(salariu_baza)
from v_angajati
where A.cod_dep=cod_dep
);