2.5.- El llenguatge estàndard SQL

- El SQL és un llenguatge estàndard de definició i manipulació (i consulta) de bases de dades relacionals.

- El SQL estàndard inclou:
    - característiques de l'Àlgebra Relacional.
    - característiques del Càlcul Relacional de Tuples.

- La versió que actualment es troba més estesa és el SQL2 (o SQL-92).

2.5.1.- SQL com llenguatge de definició de dades (DDL)

Instruccions del SQL per a poder definir esquemes relacionals:

- **create schema**: permet donar nom a un esquema relacional i declarar l'usuari que és el creador i propietari de l'esquema.
- **create domain**: permet definir un nou domini de dades.

ORACLE
- **create table**: defineix una taula, el seu esquema i les restriccions associades.

ORACLE
- **create view**: defineix una vista o relació derivada en l'esquema relacional.

- **create assertion**: permet definir restriccions d'integritat generals.

ORACLE
- **grant**: permet definir autorització d'operacions sobre objectes de la BD.

Totes aquestes instruccions tenen associada l'operació inversa (DROP / REVOKE) i modificació (ALTER).

2.5.1.1.- Definició de l'Esquema (SQL)

create schema [*esquema*] [authorization *usuari*]

[llista_element_esquema];

Un element d'esquema pot ser un dels següents:

- definició de domini.
- definició de taula.
- definició de vista.
- definició de restricció.
- definició de privilegi.

Eliminació de la definició d'un esquema relacional:

drop schema *esquema* {restrict | cascade};

2.5.1.2.- Definició de dominis (SQL)

create domain *domini* [as] tipus_dades

[default {literal | funció_sistema | null }]

[definició_restricció_domini];

Funcions del sistema:

- user
- current_user
- session_user
- current_date
- current_time
- current_timestamp.

2.5.1.2.- Definició de dominis (SQL)

A un domini se li pot associar un conjunt de restriccions:

[constraint *restricció*]

check (expressió_condicional)

[not] deferrable

- *expressió_condicional* permet expressar qualsevol condició que deu complir sempre el domini (deu ser CERTA o INDEFINIDA)
- **deferrable** indica que el sistema ha de comprovar la restricció en finalitzar la transacció activa.
- **Not deferrable** indica que el sistema ha de comprovar la restricció després de cada operació d'actualització a la base de dades.

2.5.1.2.- definició de Dominios (SQL). Exemple

CREATE DOMAIN *angle* AS FLOAT

DEFAULT 0

CHECK (VALUE >= 0 AND VALUE < 360)

NOT DEFERRABLE;

Eliminació d'un domini:

drop domain *domini* [restrict | cascade]

2.5.1.3.- Definició de taules (SQL).

CREATE TABLE taula

comallista_definició_columna

[comallista_definició_restricció_taula];

La definició d'una columna d'una taula es realitza com segueix:

columna {tipus_dades | *domini*}

[default {literal | funció_sistema | null }]

[llista_definició_restricció_columna]

Les restriccions que es poden definir sobre les columnes són les següents:

- not null: restricció de valor no nul.
- definicions de restriccions de CP, UNI, CAl d'una sola columna.
- definició de restriccions generals amb la clàusula check.

2.5.1.3.- Definició de taules (SQL).

La clàusula per a definir restriccions de taula és la següent:

[constraint *restricció*]
{ primary key (*comallista_columna*)
| unique (*comallista_columna*)
| foreign key (*comallista_columna*)
references *taula*[(*comallista_columna*)]
[match {full | partial}] * NO ORACLE
[on update [cascade | * NO ORACLE
set null | set default | no action]] * NO ORACLE
[on delete [cascade |
set null | set default | no action]] * NO ORACLE
| check *expressió_condicional* }
[*comprovació_restricció*]

Valor per defecte: l'operació no es pot fer

- deu ser certa o INDEFINIDA.
- no pot incloure subconsultes ni referències a altres taules.

2.5.1.3.- Exemple: Proveïdor-Peces-Subministre

d_cod_peça: tira(4)
d_cod_proy: tira(4)
d_dni: enter (positiu)

Proveïdor(dni: d_dni, nom: tira(40), adreça: tira(25), ciutat: tira(30))
CP: {dni}
VNN: {nom}

Peça(codi: d_cod_peça, desc: tira(40), color: tira(20), pes: real)
CP: {codi}

Subministre(dni: d_dni, codi: tira(4), preu: real)
CP: {dni, codi}
CAl: {dni} → proveïdor
CAl: {codi} → peça

Restriccions d'integritat:

R1) Px: peça ∀Px: peça (Px.color='roig' → Px.pes>100)

R2) Px: peça, Sx: Subministre ∀Px: peça (∃Sx: Subministre (Sx.codi=Px.codi))

2.5.1.3.- Exemple: Proveïdor-Peces-Subministre (SQL)

```
create schema Magatzem
    authorization pep
    create domain d_cod_peça as char(4)
    create domain d_cod_proy as char(4)
    create domain d_dni as integer check value>0
    create table proveïdor      ( dni          d_dni     primary key,
                                  nom          varchar(40)      not null,
                                  adreça       char(25),
                                  ciutat       char(30)                  )
    create table peça           ( codi         d_cod_peça       primary key,
                                  desc         varchar(40),
                                  color        char(20),
                                  pes          float,
                                  constraint r1 check (color<>'roig' or pes>100))
    create table Subministre    ( dni          d_dni,
                                   codi        d_cod_peça       references peça,
                                  preu         float,
                                  primary key (dni, codi),
                                  foreign key (dni) references Proveïdor(dni) );
```

⇐ R1

¿i R2?

2.5.1.3.- Definició de taules (SQL). Clàusula MATCH

- completa (**match full**): en cada tupla de *R* la clau aliena *CA* té el valor nul o no el té, en cadascuna de els seus columnes. En el segon cas, ha d'existir una fila en la taula *S* el valor del qual siga idèntic en les columnes de *CU*.

- parcial (**match partial**): en cada tupla de *R* la clau aliena *CA* té el valor nul en cadascuna de les seues columnes, o ha d'existir una fila en la taula *S*, de forma que per a les columnes de la clau aliena *CA* que no tenen valor nul, el valor en les columnes corresponents de *CU* és idèntic.

- dèbil (**no s'inclou clàusula match**): en cada tupla de *R* si la clau aliena *CA* no té el valor nul, en cadascuna de les seues columnes, ha d'existir una fila en la taula *S* tal que el valor en *CU* coincidisca en totes les columnes.

ORACLE

2.5.1.3.- Modificació de definició de taules (SQL).

Per a modificar la definició d'una taula:

alter table *taula_base*

{add [column] definició_columna

| alter [column] *columna*

{set default {literal | funció_sistema | null }

| drop default}

| drop [column] *columna* {restrict | cascade} };

En ORACLE canvien algunes coses

Per a eliminar una taula de l'esquema relacional:

drop table *taula_base* {restrict | cascade};

En ORACLE és CASCADE CONSTRAINTS

2.5.1.4.- Definició de restriccions (SQL)

create assertion *restricció*

check (expressió_condicional)

[comprovació_restricció];

La condició deu ser certa.

2.5.1.4.- Exemple: Proveïdor-Peces-Subministre (SQL)

La restricció R2 :

R2) Px: peça, Sx: Subministre ∀Px : peça (∃Sx : Subministre(Sx) (Sx.codi=Px.codi))

es defineix mitjançant una restricció general:

```
create assertion R2 check
not exists(select * from peça P
            where not exists(select *
                              from Subministre S
                              where P.codi=S.codi));
```

Eliminació d'una restricció

DROP ASSERTION *restricció*

2.5.2.- SQL com a llenguatge de manipulació de dades.

- El SQL com a llenguatge de manipulació de dades incorpora:
  - La sentència de consulta SELECT: integració de les perspectives lògica i algebraica.
  - Les sentències d'actualització de dades: INSERT, DELETE i UPDATE.

2.5.2.1- La sentència SELECT

SELECT

- Permet recuperar informació emmagatzemada en una base de dades.

- La seua sintaxi és:

5 **select** [**all** | **distinct**] *comallista_ítem_seleccionat* | *

1 **from** taula

2 [**where** *expressió_condicional*]

3 [**group by** *comallista_col*]

4 [**having** *expressió_condicional*]

6 [**order by** *comallista_referència_col*]

2.5.2.1- La sentència SELECT

3 **select** R1X.A, R2X.B, ... , RnX.AA

1 **from** R1 [AS] R1X, R2 [AS] R2X, ... , Rn [AS] RnX

2 [**where** *F(R1X, R2X, ..., RnX)*]

on:

- R1, R2, .., Rn són relacions.
- A, B, ..., AA són atributs de les corresponents relacions.
- R1X, R2X, ..., RnX són noms alternatius (àlies).
- F(R1X, R2X, ..., RnX) és una condició.

El resultat és una relació formada pels atributs A, BB, ..., AA de les tuples de les relacions R1, R2, ..., Rn per a les quals F és certa.

2.5.2.1- La sentència SELECT.

REANOMENAR

- Per a realitzar reanomenaments en SQL s'utilitza la paraula reservada AS.
- Permet el reanomenament d'una relació així com de tots els seus atributs (ULL: no canvia l'esquema de la relació).

EXEMPLES:

Jugador(nom:varchar, edat: number, país:varchar)

Jugador AS Tenista ⟹ reanomena la relació *Jugador*

Jugador AS T(nom, ed, pa) ⟹ reanomena la relació *Jugador* i tots els seus atributs.

2.5.2.1- La sentència SELECT.

REANOMENAR *(Cont)*

- La paraula reservada AS és opcional.
- En el cas de que una consulta faça referència dues o més vegades a una mateixa taula, resulta imprescindible realitzar un reanomenament.

Exemple:

Jugador(dni:number, nom:varchar, edat: number, país:varchar)

CP:{dni}

Obtén la llista de parells de noms de jugadors del mateix país:

```
Select J1.nom, J2.nom
from Jugador AS J1, Jugador AS J2
where J1.país = J2.país and J1.dni < J2.dni;
```

2.5.2.1- La sentència SELECT: l'aproximació lògica.

3 **select** R1X.A, R2X.B, ... , RnX.AA

1 **from** R1 [AS] R1X, R2 [AS] R2X, ... , Rn [AS] RnX

2 [**where** *F(R1X, R2X, ..., RnX)*]

on:

- En el SELECT s'indiquen els atributs que es desitgen consultar.
- En la component FROM es declaren variables de tipus tupla.
- WHERE és una fórmula lògica en la qual les úniques variables lliures són les declarades en el FROM.
- La fórmula del WHERE es construeix seguint la sintaxi usual dels llenguatges de 1er ordre.

2.5.2.1- La sentència SELECT: l'aproximació lògica.

FORMALITZACIÓ (SINTAXI):

Fórmules de la clàusula WHERE.

Una **condició** és una expressió que pot ser:

- IS NULL (RX.Ai)
- RX.Ai α SX.Aj
- RX.Ai α *a*

on:

- α és un operador de comparació ($<$, $>$, $\leq$, $\geq$, $=$, $\neq$).
- Ai i Aj són noms d'atribut de les relacions sobre les quals s'han definit les variables RX i SX.
- *a* és un valor del domini associat a l'atribut RX.Ai (excepte el nul).

2.5.2.1- La sentència SELECT: l'aproximació lògica.

Amb açò, les **fórmules** es construeixen aplicant les següents regles:

- Tota condició és una fórmula.
- Si ***F*** és una fórmula, aleshores *(**F**)* i *NOT **F*** són fórmules.
- Si ***F*** i ***G*** són fórmules, aleshores també ho són ***F OR G***, ***F*** AND ***G***.
- Si ***S*** és una sentència SELECT, aleshores EXISTS(S) és una fórmula.
- res més no és una fórmula.

2.5.2.1- La sentència SELECT: l'aproximació lògica.

3 **select** R1X.A, R2X.B, ... , RnX.AA

1 **from** R1 [AS] R1X, R2 [AS] R2X, ... , Rn [AS] RnX

2 [**where** *F(R1X, R2X, ..., RnX)*]

La sentència SELECT retorna una relació en la que cada tupla de la relació es forma pels valors dels atributs R1X.A, R2X.B, ... , RnX.AA de tal forma que:

- Aquests valors apareixen en les variables R1X, R2X, ..., RnX.
- Per tant, aquests valors apareixen en les extensions de les relacions R1, R2, ..., Rn.
- Aquests valors fan certa la fórmula *F(R1X, R2X, ..., RnX)*.

2.5.2.1- La sentència SELECT: l'aproximació lògica.

AVALUACIÓ DE FÓRMULES (SEMÀNTICA).

Valor de veritat d'una condició:

- Si F és de la forma RX.Ai α SX.Aj aleshores F s'avalua a indefinit si almenys un atribut Ai o Aj té el valor nul en la tupla assignada a RX o a SX, en cas contrari s'avalua al valor de certesa de la condició.
- Si F és de la forma RX.Ai α *a* aleshores F s'avalua a indefinit si Ai té valor nul en la tupla assignada a RX, en cas contrari s'avalua al valor de certesa de la comparació.
- Si F és de la forma IS NULL(RX.Ai) aleshores F s'avalua a cert si Ai té el valor nul per a la tupla assignada a RX, en cas contrari s'avalua a fals.

2.5.2.1- La sentència SELECT: l'aproximació lògica.

Valor de veritat d'una fórmula:

1) Siga *F* una condició, aleshores el seu valor de veritat és el de la condició.

2) Si *F* és de la forma *(G)*, *F* s'avalua al valor de certesa de *G*.

3) Si *F* és d'una de les següents formes NOT *G*, *G* AND *H* ó *G* OR *H* on *G* i *H* són fórmules, aleshores *F* s'avalua d'acord a les següents taules de veritat:

2.5.2.1- La sentència SELECT: l'aproximació lògica.

G	H	F =G AND H	F = G OR H
fals	fals	fals	fals
indefinit	fals	fals	indefinit
cert	fals	fals	cert
fals	indefinit	fals	indefinit
indefinit	indefinit	indefinit	indefinit
cert	indefinit	indefinit	cert
fals	cert	fals	cert
indefinit	cert	indefinit	cert
cert	cert	cert	cert

G	F = NOT G
fals	cert
indefinit	indefinit
cert	fals

2.5.2.1- La sentència SELECT: l'aproximació lògica.

4) Si *F* és de la forma:

EXISTS(select *

from R1 [AS] R1X, R2 [AS] R2X, ... , Rn [AS] RnX

[**where** *G(R1X, R2X, ..., RnX)*])

aleshores F s'avalua a cert si existeixen valors de les variables R1X, ..., RnX de les extensions de R1, ..., Rn per als quals G s'avalua a cert, en cas contrari s'avalua a fals.

2.5.2.1- La sentència SELECT: l'aproximació lògica.

Exemple:

RIU(rcod:dom_rcod, nom:dom_nom)

PROVÍNCIA(pcod:dom_pcod, nom:dom_nom)

PASSA_PER(pcod:dom_pcod, rcod:dom_rcod)

Consulta1: "províncies per les quals passa el riu de codi r1".

lògica de 1er ordre: Provincia(x,y) ^ Passa_per(x, 'r1')

Variables tupla: PX:Província | ∃PPX:Passa_per (PPX.pcod = PX.pcod ^ PPX.rcod = 'r1')

Clàusula SELECT:

```
SELECT PX.pcod, PX.nom
FROM Província PX, Passa_per PPX
WHERE EXISTS(SELECT *
             FROM Passa_per PPX
             WHERE PPX.pcod = PX.pcod AND PPX.rcod = 'r1')
```

2.5.2.1- La sentència SELECT: l'aproximació lògica.

Exemple:

RÍO(rcod:dom_rcod, nom:dom_nom)

PROVINCIA(pcod:dom_pcod, nom:dom_nom)

PASA_POR(pcod:dom_pcod, rcod:dom_rcod)

Consulta2: “províncies per les quals no passa cap riu”.

Lògica de 1er ordre: Província(x,y) ^ ¬∃z Passa_per(x, z)

Variables tupla: PX:Província | ¬∃PPX:Passa_per (PPX.pcod = PX.pcod)

Clàusula SELECT:

```
SELECT *
FROM província PX
WHERE NOT EXISTS(SELECT *
                 FROM Passa_per PPX
                 WHERE PPX.pcod = PX.pcod)
```

2.5.2.1- La sentència SELECT: l'aproximació lògica.

La sintaxi del quantificador existencial en el llenguatge SQL:

EXISTS(SELECT *

FROM R1 R1X, R2 R2X, ..., Rn RnX

WHERE F(R1X, R2X, ..., RnX))

equival a la fórmula $\exists$R1X:R1($\exists$R2X:R2 ...($\exists$RnX:Rn (F(R1X, R2X, ..., RnX))...)

En SQL no existeix el quantificador universal, s'utilitza l'existencial en el seu lloc mitjançant la conversió: $\forall x\ F(x) \equiv \neg\exists x\ (\neg\ F(x))$

2.5.2.1- La sentència SELECT: l'aproximació lògica.

Exemple:

RIU(rcod:dom_rcod, nom:dom_nom)

PROVÍNCIA(pcod:dom_pcod, nom:dom_nom)

PASSA_PER(pcod:dom_pcod, rcod:dom_rcod)

Consulta3: "Obteniu els rius que passen por totes les províncies".

Variables tupla:

RX:Riu|∀PX:Província (∃PPX:Passa_per (PPX.pcod=PX.pcod ^ PPX.rcod=RX.rcod))

RX:Riu|¬∃PX:Província (¬∃PPX:Passa_per (PPX.pcod=PX.pcod ^ PPX.rcod=RX.rcod)

Clàusula SELECT:

```
SELECT * FROM riu RX
WHERE NOT EXISTS (SELECT * FROM Província PX
                  WHERE NOT EXISTS(SELECT * FROM Passa_per PPX
                                   WHERE PPX.pcod = PX.pcod AND
                                         PPX.rcod = RX.rcod))
```

2.5.2.1- La sentència SELECT: l'aproximació algebraica.

UNIÓ

- Permet fusionar el contingut de dues relacions (o resultats de consultes) en una única taula.
- La correcta execució de l'operació d'unió requereix que les dues relacions que s'uneixen siguen compatibles.

Exemple:

cuiner(nom:varchar, edat: number, país:varchar)

cambrer(nom:varchar, edat: number, país:varchar)

Obtín la llista de treballadors majors d'edat del restaurant:

```
Select nom from cuiner where edat >= 18
UNION
Select nom from cambrer where edat >= 18;
```

2.5.2.1- La sentència SELECT: l'aproximació algebraica.

DIFERÈNCIA

- La paraula reservada en SQL per a realitzar diferències entre relacions és EXCEPT.
- La correcta execució de l'operació de diferència requereix que les dues relacions siguen compatibles.

Exemple:

cuiner(nom:varchar, edat: number, país:varchar)

cambrer(nom:varchar, edat: number, país:varchar)

Obtín la llista de treballadors que treballen únicament com cuiners en el restaurant:

1. **Select * from** (cuiner **except** cambrer)
2. cuiner **except** cambrer

2.5.2.1- La sentència SELECT: l'aproximació algebraica.

INTERSECCIÓ

- La paraula reservada en SQL per a realitzar diferències entre relacions és INTERSECT.
- La correcta execució de l'operació de diferència requereix que les dues relacions que intersecten siguen compatibles.

Exemple:

cuiner(nom:varchar, edat: number, país:varchar)

cambrer(nom:varchar, edat: number, país:varchar)

Obtín la llista de treballadors que treballen com a cuiners i cambrers en el restaurant:

1. **Select * from** (cuiner **intersect** cambrer)
2. cuiner **intersect** cambrer

2.5.2.1- La sentència SELECT: l'aproximació algebraica.

PRODUCTE CARTESIÀ

- La correcta execució de l'operació de producte cartesià requereix que les relacions que intervenen tinguen diferents noms.
- En SQL, el producte cartesià de relacions s'aplica afegint les relacions, separades per comes, dins de la clàusula FROM.

Exemple:

Equip1(nom:varchar, edat: number, país:varchar)

Equip2(nom:varchar, edat: number, país:varchar)

Obtín totes les possibles combinacions de jugadors de l'equip 1 amb jugadors de l'equip 2:

Select * from Equip1, Equip2
Select * from Equip1 **CROSS JOIN** Equip2

Obtín parells de jugadors de l'Equip1 del mateix país:

Select * from Equip1 e1, Equipo1 e2 **where** e1.país = e2.país and e1.edat < e2.edat

2.5.2.1- La sentència SELECT: l'aproximació algebraica.

PROJECCIÓ

- Per a projectar basta amb escriure el nom dels atributs que es desitgen visualitzar dins de la clàusula SELECT separats per comes.
- Els atributs projectats es poden reanomenar utilitzant la clàusula AS.

Exemple:

cuiner(nom:varchar, edat: number, país:varchar)

Obtín el nom dels cuiners del restaurant:

Select nom **from** cuiner

2.5.2.1- La sentència SELECT: l'aproximació algebraica.

CONCATENACIÓ

- Corresponen a variants de l'operador concatenació de l'àlgebra relacional.
- Hi ha dos tipus bàsics de concatenació en SQL: Interna i Externa.
- Concatenació interna:

  referència_taula [natural] [inner] join *referència_taula*
  [on *expressió_condicional* | using (*comallista_columna*)]

- concatenació externa:

  referència_taula [natural]
  {left [outer] | right [outer] | full [outer]} JOIN *referència_taula*
  [on *expressió_condicional* | using (*comallista_columna*)]

2.5.2.1- La sentència SELECT: l'aproximació algebraica.

CONCATENACIÓ *(Cont.)*

EXEMPLES: concatenació interna.

referència_taula [natural] [inner] join *referència_taula*

[on *expressió_condicional* | using (*comallista_columna*)]

PERSONA(nif: dom_nif, nom: dom_nom, edat: dom_edat)

HABITATGE(cod_hab: dom_cod, prop: dom_nif, dir: dom_dir, num_camb: dom_num)

- Obteniu un llistat en el qual aparega cada habitatge associat amb el seu propietari:

1. PERSONA inner join HABITATGE on PERSONA.nif = HABITATGE.prop
2. PERSONA natural inner join HABITATGE AS V(ch, nif, dir, nc)
3. SELECT * FROM PERSONA, HABITATGE WHERE nif = prop

2.5.2.1- La sentència SELECT: l'aproximació algebraica.

CONCATENACIÓ *(Cont.)*

EXEMPLES: concatenació externa.

referència_taula [natural]

{left [outer] | right [outer] | full [outer]} JOIN *referència_taula*

[on *expressió_condicional* | using (*comallista_columna*)]

PERSONA(nif: dom_nif, nom: dom_nom, edat: dom_edat)

HABITATGE(cod_hab: dom_cod, nif: dom_nif, dir: dom_dir, num_camb: dom_num)

- Obteniu un llistat en el qual aparega cada habitatge associat amb el seu propietari:

1. PERSONA natural left join habitatge ⟹ apareixen tots els propietaris
2. PERSONA natural right join habitatge ⟹ apareixen tots els habitatges
3. PERSONA natural full join habitatge ⟹ apareixen tots els habitatges i tots els propietaris

2.5.2.1- La sentència SELECT: l'aproximació algebraica.

CONCATENACIÓ *(Cont.)*

- concatenació unió

... FROM T1 UNION JOIN T2 ≡ ... FROM

select t1.*, null, null, ..., null from t1

union all

select null, null, ..., null, t2.* from t2

2.5.2.1- La sentència SELECT: l'aproximació algebraica.

SELECCIÓ

- L'expressió de l'àlgebra relacional:

R **ON** F(Ai, Aj, Ak, ...)

és equivalent a l'expressió en SQL:

SELECT * **FROM** R **WHERE** F(R.Ai, R.Aj, R.Ak, ...)

- En el cas de que s'incloguen diverses relacions en la clàusula FROM del SELECT:

SELECT * **FROM** R1, R2, ..., Rn **WHERE** F(R1.Ai, ..., Rn.Zk)

El seu equivalent en àlgebra relacional seria:

R1 x R2 x ... x Rn **ON** F(R1.Ai, ..., Rn.Zk)

2.5.2.1- La sentència SELECT: l'aproximació algebraica.

Operador	Àlgebra Relacional	SQL
Selecció	R ON F	SELECT ... FROM R WHERE F
Projecció	R [A_i, A_j ..., A_k]	SELECT A_i, A_j ..., A_k FROM R
Producte Cartesià	R_1 x R_2, ... x R_n	SELECT ... FROM R_1, R_2, ..., R_n, o SELECT...FROM R_1 CROSS JOIN R_2, ..., CROSS JOIN R_n
Concatenació	$R_1 \bowtie R_2$	SELECT... FROM R_1 NATURAL JOIN R_2
Unió	$R_1 \cup R_2$	SELECT * FROM R_1 UNION SELECT * FROM R_2
Diferència	R_1 - R_2	SELECT * FROM R_1 EXCEPT SELECT * FROM R_2
Intersecció	$R_1 \cap R_2$	SELECT * FROM R_1 INTERSECT SELECT * FROM R_2

2.5.2.1- La sentència SELECT: l'aproximació algebraica.

Exemple:

RIU(rcod:dom_rcod, nom:dom_nom)

PROVÍNCIA(pcod:dom_pcod, nom:dom_nom)

PASSA_PER(pcod:dom_pcod, rcod:dom_rcod)

Consulta2: “províncies por les quals no passa cap riu”.

Àlgebra Relacional: Provincia[pcod, nom] – (província ⋈ Passa_per)[pcod, nom]

SQL:

```
SELECT pcod, nom FROM província
EXCEPT
SELECT pcod, nom FROM província NATURAL JOIN Passa_per
```

2.5.2.2- SQL com a llenguatge de manipulació de dades: l'actualització.

INSERT

- Serveix per a inserir una o més tuples en una relació.

- La seua sintaxi és:

  insert into taula [(*comallista_columna*)]

  { default values | values (*comallista_àtoms*) | *expressió_taula*}

- Si no incloguem la llista de columnes (comallista_columna), s'esperaran valors per a totes les columnes en l'ordre de definició de la relació.

2.5.2.2- SQL com a llenguatge de manipulació de dades: l'actualització.

cuiner(nom:varchar, edat: number, país:varchar)

nom	edat	País
. . .	. . .	. . .

INSERT INTO cuiner

VALUES (“Pol Cotó”, 27, “França”);

2.5.2.2- SQL com a llenguatge de manipulació de dades: l'actualització.

cuiner(nom:varchar, edat: number, país:varchar)

nom	edat	País
. . .	. . .	. . .
Pol Cotó	27	França
. . .	. . .	. . .

INSERT INTO cuiner

VALUES (“Pol Cotó”, 27, “França”);

2.5.2.2- SQL com a llenguatge de manipulació de dades: l'actualització.

cuiner(nom:varchar, edat: number, país:varchar)

nom	edat	País
. . .	. . .	. . .

INSERT INTO cuiner(Edat, nom)

VALUES (27, “Pol Cotó”);

2.5.2.2- SQL com a llenguatge de manipulació de dades: l'actualització.

cuiner(nom:varchar, edat: number, país:varchar)

nom	edat	País
. . .	. . .	. . .
Pol Cotó	27	?
. . .	. . .	. . .

INSERT INTO cuiner(Edat, nom)

VALUES (27, “Pol Cotó”);

2.5.2.2- SQL com a llenguatge de manipulació de dades: l'actualització.

insert into taula [(comallista_columna)]

{ default values | values (comallista_àtoms) | expressió_taula}

- Si s'inclou l'opció *default values* s'inserirá una única fila en la taula amb els valors por defecte apropiats en cada columna (segons la definició de *taula*).
- En l'opció values(*comallista_àtoms*) els àtoms venen donats per expressions escalars.
- En l'opció *expressió_taula*, s'inserirán les files resultants de l'execució de l'expressió (SELECT).

2.5.2.2- SQL com a llenguatge de manipulació de dades: l'actualització.

cuiner(nom:varchar, edat: number, país:varchar)

nom	edat	País
. . .	. . .	. . .

Persona(nom:varchar, edat: number)

nom	edat
Paco	22
Antoni	19
Soletat	26

INSER INTO cuiner(nom, edat)

SELECT nom, edat

FROM Persona

WHERE edat > 20;

2.5.2.2- SQL com a llenguatge de manipulació de dades: l'actualització.

cuiner(nom:varchar, edat: number, país:varchar)

nom	edat	País
. . .	. . .	. . .
Paco	22	?
Soletat	26	?
. . .	. . .	. . .

INSER INTO cuiner(nom, edat)

SELECT nom, edat

FROM Persona

WHERE edat > 20;

2.5.2.2- SQL com a llenguatge de manipulació de dades: l'actualització.

UPDATE

- Serveix per a modificar els valors dels atributs d'una o més tuples seleccionades.

- La seua sintaxi és:

update *taula*

set *comallista_assignacions*

[where *expressió_condicional*]

on una *assignació* és de la forma:

columna = {default | null | *expressió_escalar*}

2.5.2.2- SQL com a llenguatge de manipulació de dades: l'actualització.

Si s'inclou la clàusula **where** només s'aplicarà a les files que facen certa la condició.

Exemple: Disminuir en una unitat l'edat dels cuiners francesos.

UPDATE cuiner **SET** edat = edat - 1

WHERE País = "França" ;

2.5.2.2- SQL com a llenguatge de manipulació de dades: l'actualització.

DELETE

- Elimina una o més tuples d'una relació.
- La seua sintaxi és:

  Delete from *taula* [where *expressió_condicional*]

- Si s'inclou la clàusula *where* s'eliminaran aquelles que facen certa la condició.

Exemple: Eliminar la informació dels cuiners menors de 18 anys.

DELETE FROM cuiner **WHERE** edat < 18;

2.6.1.- Concepte de vista.

- Una vista és una taula derivada d'altres taules (bàsiques o virtuals).
- Una vista es caracteritza perquè:
  - es considera que forma part de l'esquema extern.
  - una vista és una taula virtual (no té una correspondència a nivell físic)
  - es pot consultar com qualsevol taula bàsica.
  - les actualitzacions es transfereixen a la/les taula/es original/s (amb certes limitacions).

2.6.2.- Aplicacions de les vistes.

- per a l'especificació de taules amb informació que s'accedeix amb freqüència però no posseeix existència física:
  - informació derivada de la relació entre diverses taules.
  - informació derivada de la formació de grups de tuples (p.ex. per a l'obtenció d'estadístiques).
  - En general: informació derivada de consultes complexes a la qual s'accedeix amb freqüència.

- com a mecanisme de seguretat: creació de vistes amb, únicament, els atributs de les taules als quals es desitja permetre accedir a determinats usuaris.

- per a la creació d'esquemes externs.

2.6.3.- Vistes en SQL.

- La sintaxi per a la creació de vistes en SQL és la següent:

CREATE | REPLACE VIEW *vista* [(*comallista_columna*)]

AS *expressió_taula* [with check option]

on:

– CREATE VIEW és l'ordre que permet la creació de la vista.

– *vista* és el nom de la taula virtual que es crearà.

– (*comallista_columna*) són els noms dels atributs de la taula i és opcional:

  - Si no s'especifica, el nom coincideix amb el nom dels atributs resultants en *expressió_taula*.
  - és obligatori si algun atribut d'*expressió_taula* és el resultat d'una funció d'agregació o una operació aritmètica.

2.6.3.- Vistes en SQL.

- La sintaxi per a la creació de vistes en SQL és la següent:

  CREATE | REPLACE VIEW *vista* [(*comallista_columna*)]

  AS *expressió_taula* [with check option]

on:

– *expressió_taula* és una consulta SQL el resultat del qual serà el contingut de la vista.

– WITH CHECK OPTION és opcional i es deu incloure si es desitja actualitzar la vista d'una manera íntegra.

– per a l'eliminació d'una vista s'utilitza la instrucció:

  – DROP VIEW *vista* [restrict | cascade];

2.6.3.- Vistes en SQL (Exemples).

- Donada la següent relació d'una base de dades:

  cuiner(nom:varchar, edat: number, país:varchar)

Obtín una vista amb, únicament, els cuiners francesos:

```
CREATE VIEW Francesos AS
    SELECT * FROM cuiner WH
    WITH CHECK OPTION
```

El Check Option impedeix que es puguen afegir cuiners que no siguen francesos

Obtín una vista amb l'edat mitjana dels cuiners agrupats por país:

```
CREATE VIEW Estudi(país, edad_mitjana) AS
    SELECT país, AVG(edad) FROM cuiner GROUP BY país
```

2.6.3.- Vistes en SQL (Vistes Actualitzables).

Motius pels quals una vista NO és actualitzable:

- conté operadors conjuntistes (UNION, INTERSECT,…).
- l'operador DISTINCT
- funcions agregades (SUM, AVG, ..)
- la clàusula GROUP BY

2.6.3.- Vistes en SQL (Vistes Actualitzables).

Vista sobre una taula bàsica:

- el sistema traduirà l'actualització sobre la vista en una operació d'actualització sobre la relació bàsica.

- sempre que no es viole cap restricció d'integritat definida sobre la relació.

2.6.3.- Vistes en SQL (Vistes Actualitzables).

Vista sobre una concatenació de relacions:

- L'actualització només pot modificar una de les taules bàsiques
- L'actualització modificarà la relació bàsica que complisca la propietat de *conservació de la clau* (aquella relació tal que la seua clau primària podria ser també clau de la vista)
- L'actualització no es realitzarà si viola alguna de les restriccions definides sobre la relació bàsica que es pensava actualitzar

2.6.3.- Vistes en SQL (Vistes Actualitzables).

Exemple:

- Donades les següents relacions:

Persona(nif: dom_nif, nom: dom_nom, edat: dom_edat)
CP: {nif}

Habitatge(cod_hab: dom_cod, nif: dom_nif, adr: dom_adr, num_camb: dom_num)
CP: {cod_hab} CAl: {nif} → Persona

- Donada la següent vista definida sobre aquestes relacions:

```
CREATE VIEW Tot_Habitatge AS
    SELECT * FROM Persona NATURAL JOIN habitatge
```

Podré modificar l'adreça d'un habitatge en Tot_Habitatge?

Sí, la CP de habitatge podria funcionar com CP de Tot_Habitatge

Podré modificar el nom del propietari d'un habitatge?

No, l'actualització real és ambígua

2.7.1.- Concepte de disparador.

Són regles que especifiquen accions que són activades automàticament per determinats esdeveniments.

2.7.2.- Regles Esdeveniment-Condició-Acció.

Forma d'una regla d'activitat:

esdeveniment - condició - acció

acció que el sistema executa com resposta a l'ocurrència d'un *esdeveniment* quan certa *condició* se satisfà:

- *esdeveniment*: operació d'actualització
- *condició*: expressió lògica del SQL. Només s'executarà l'acció si aquesta condició existeix i és vertadera.
- *acció*: procediment escrit en un llenguatge de programació que inclou instruccions de manipulació de la BD.

2.7.3.- Aplicacions dels Disparadors.

Defineix el comportament actiu del sistema. Aplicacions:

- comprovació de restriccions de integritat
- restauració de la consistència
- definició de regles de funcionament de l'organització
- manteniment d'informació derivada

2.7.4.- Disparadors en SQL.

definició_regla::=
{CREATE | REPLACE} TRIGGER nom_regla
{BEFORE | AFTER | INSTEAD OF} ***esdeveniment*** [*disjunció_esdeveniments*]
ON {nom_relació | nom_vista}
[[REFERENCING OLD AS nom_referència
[NEW AS nom_referència]]
[FOR EACH {ROW | STATEMENT} [WHEN (***condició***)]]
bloc PL/SQL

disjunció_esdeveniment ::= OR *esdeveniment* [*disjunció_esdeveniments*]

esdeveniment ::= INSERT | DELETE | UPDATE [OF comallista_nom_atribut]

2.7.4.- Disparadors en SQL.

esdeveniments

{BEFORE | AFTER | INSTEAD OF} ***esdeveniment*** [*disjunció_esdeveniments*]
ON {nom_relació | nom_vista}

disjunció_esdeveniments ::= OR *esdeveniment* [*disjunció_esdeveniments*]

esdeveniment ::=
INSERT | DELETE | UPDATE [OF comallista_nom_atribut]

2.7.4.- Disparadors en SQL.

Esdeveniments

Parametrització d'esdeveniments:

- Els esdeveniments de les regles FOR EACH ROW estan parametritzats
- Parametrització implícita:
  - esdeveniment INSERT o DELETE: *n* (*n* grau de la relació)
  - esdeveniment UPDATE: 2**n*
- Nom de paràmetres:
  - esdeveniment INSERT: *NEW*
  - esdeveniment DELETE: *OLD*
  - esdeveniment UPDATE: *OLD* i *NEW*
- es poden usar en la *condició de la regla*
- es poden usar en el bloc PL/SQL

2.7.4.- Disparadors en SQL.

	FOR EACH STATEMENT	FOR EACH ROW
BEFORE	La regla s'executa una vegada abans de l'execució de l'operació d'actualització	La regla s'executa una vegada abans de l'actualització de cada tupla afectada per l'operació d'actualització
AFTER	La regla s'executa una vegada després de l'execució de l'operació d'actualització	La regla s'executa una vegada després de l'actualització de cada tupla afectada per l'operació d'actualització

2.7.4.- Disparadors en SQL.

CONDICIONS

WHEN (condició)

- expressió lògica de sintaxi similar a la condició de la clàusula WHERE de la instrucció SELECT
- no pot contenir subconsultes ni funcions agregades
- només es pot fer referència als paràmetres de l'esdeveniment

2.7.4.- Disparadors en SQL.

accions

bloc PL/SQL

- bloc escrit en el llenguatge de programació d'Oracle PL/SQL
- sentències de manipulació de la BD: INSERT, DELETE, UPDATE, SELECT ... INTO ...
- sentències de programa: assignació, selecció, iteració
- sentències de gestió d'errors
- sentències d'entrada/eixida

2.7.4.- Disparadors en SQL.

Llenguatge de regles:

- Creació: CREATE TRIGGER nom_regla ...
- Eliminació: DROP TRIGGER nom_regla
- Modificació: REPLACE TRIGGER nom_regla ...
- Recompilació: ALTER TRIGGER nom_regla COMPILE
- Des/Habilitar regla: ALTER TRIGGER nom_regla [ENABLE | DISABLE]
- Des/Habilitar totes les regles sobre una relació:

  ALTER TABLE nom_relació [{ENABLE | DISABLE} ALL TRIGGERS]

2.7.4.- Disparadors en SQL (Exemple).

La restricció R2 :

R2) Px: Peça, Sx: Subministre ∀Px : Peça (∃Sx : Subministre (Sx.codi=Px.codi))

es defineix mitjançant una restricció general:

```
create assertion R2 check
not exists(select * from Peça P
            where not exists(select *
                              from Subministre S
                              where P.codi=S.codi));
```

Com controlar la restricció mitjançant regles d'activitat?

2.7.4.- Disparadors en SQL (Exemple).

Detectar quins esdeveniments poden afectar la R.I.:

TAULA,	OPERACIÓ,	ATRIBUT
Subministre,	Esborrat,	-
Subministre,	Modificació,	codi
Peça,	Inserció,	-

Construir Triggers per a controlar aquests esdeveniments.

2.7.4.- Disparadors en SQL (Exemple).

```
CREATE TRIGGER T1
AFTER DELETE ON Subministre OR UPDATE OF codi ON Subministre
FOR EACH ROW
DECLARE
    N: NUMBER;
BEGIN
    SELECT COUNT(*) INTO N
        FROM Subministre S
        WHERE :old.codi = S.codi;
    IF N=0 THEN
          RAISE_APPLICATION_ERROR(-20000, 'No es pot
             esborrar el subministre, perquè la peça es quedaria sense subministres.');
    END IF;
END;
```

2.7.4.- Disparadors en SQL (Exemple).

```
CREATE TRIGGER T2
AFTER INSERT ON peça
FOR EACH ROW
DECLARE    N: NUMBER;
BEGIN
     SELECT COUNT(*) INTO N
          FROM Subministre S  WHERE :new.codi = S.codi;
     IF N=0 THEN
          RAISE_APPLICATION_ERROR(-20000, 'No es pot
               inserir una peça, perquè la peça no té subministres.
               Cree les dues tuples (la de peça i la de subministre)
               dins d'una transacció deshabilitant aquest trigger.');
     END IF;
END;
```

2.8.- Limitacions del model relacional.

- Els models de dades tradicionals (relacional, jeràrquic i xarxa) han tingut èxit en aplicacions tradicionals de negocis.

- Els models tradicionals presenten deficiències en aplicacions:
  - de disseny i fabricació en enginyeria (CAD/CAM/CIM),
  - experiments científics,
  - telecomunicacions,
  - sistemes d'informació geogràfica, i
  - multimèdia,
  - Sistemes d'informàció estratègica (magatzem de dades).

2.8.- Limitacions del model relacional.

- Requisits i característiques de les noves aplicacions:
  - estructures més complexes per als objectes,
  - transaccions de major duració,
  - nous tipus de dades per a emmagatzemar imatges o elements de text grans i
  - la necessitat de definir operacions no estàndard específiques per a les aplicacions.

- Evolució de les bases de dades relacionals:
  - bases de dades deductives,
  - bases de dades actives,
  - bases de dades orientades a objectes,
  - bases de dades objecte-relacionals (SQL3),
  - magatzem de dades.