Mariona Farré
2023/24

Disseny de Bases de Dades:

Tema 1: Àlgebra relacional i SQL **2**
Laboratori: 2
Materialització: 4
Laboratori: 4
Tema 2: Modelat Conceptual **7**
● Multiplicitats: 8
● Claus: 9
● Especialitzacions: 9
● Anomalies 10
Dependència funcional: 10
Formes normals: 11
Laboratori: 14
Laboratori: 17
Tema 3: índexs, Transaccions Disparadors **19**
Índexs: 19
Arbres B+: 19
Arbre B i B+: 20
CLAUS: 21
Transaccions 22
Coherencia: 22
ACID: 22
SERIAL: 22
SERIALIZABLE: 23
REPEATABLE READ: 23
READ UNCOMMITTED: 23
READ COMMITTED: 23
Locks: 24
Disparadors 24
Tema 4: Procés de consultes SQL **25**
SELECT: 25
WHERE: 25
JOIN: 26
Tema 5: Principis de l'administrador de BD **27**
Tema 6: Bases de Dades NO SQL **28**
Teorema CAP: 28
Bases de Dades NoSQL 29

Tema 1: Àlgebra relacional i SQL
1/30

Conjunt: cardinalitat, subconjunt
Operacions entre conjunts:

- unió
- intersecció
- complementari
- producte cartesià de dos conjunts
- producte cartesià d'un nombre il·limitat de conjunts
- tuples (cada element d'un producte cartesià)

Relacions (subconjunts d'un producte cartesià):

- relació entre dos conjunts (o correspondencia)
- aplicació (o funció): relació en que cada element del primer
- conjunt està relacionat amb únic element del segon
- composició de correspondències
- relació entre un número ilimitat de conjunts

Laboratori:

sqlite3
.open set theory.bd
.database
.fullschema

sqlite3 set_theory.db
.fullschema
//instruccions//

.headers on → veuen headers bd
. tables → taules q tens
.SELECT * FROM x; → veure tot lo de les taules q tens x..x

SELECT p.q.r FROM a,b,c; → producte cartesià tots conjunts
SELECT COUNT(*) FROM a,b,c → num total elements

SELECT p FROM a UNION SELECT r FROM c; → unir selecció p i r de les diferents bd
SELECT p FROM a INTERSECT SELECT r FROM c; →intersecció entre a i c
SELECT p FROM a EXCEPT SELECT r FROM c; → complementari (diferències) entre a i c

SELECT p,q FROM a,b; → producte cartesià de a,b - totes les relacions possibles
SELECT * FROM a,b WHERE q-p=101; → subconjunt de possibles parelles q compleix condicio
SELECT * FROM a,c WHERE p>r; → Relació binaria(2) subconjunt a i c q compleixi condicio
SELECT * FROM a,b,c WHERE q-p =101 AND q-p-r=100; →Relació ternària(3) 2 condicions

Les bd relacionals s'organitzen guardant info taules: contenen registres (files) i atributs (col)

- relacions son les taules de les bd
- tuples corresponen als registres/files d'una taula
- components/atributs tupla corresponent a camps/columnes d'un registre

noms son components de les relacions

Condicions:

- Condició simple: igualtat, desigualtat o ordre
  - entre components diferents de la mateixa relació
  - entre una component i una constant
  - entre components de dos relacions
  - entre expressions aritmètiques que utilitzin aquestes components
- Condició composta:
  - combinació de condicions simples mitjançant conjuncions AND i/o disjuncions OR.

Operacions relacionals:

- Selecció WHERE
  - R'=σC (R)
    R' nova relació, conté tuples de R que compleixen condició C

- Projecció SELECT
  - R'= πA (R)
    R' nova relació conté una tupla per cada tupla de R
    De cada tupla només conserven els atributs de la llista A

→ Per evitar possibles repeticions de tuples utilitzar: DISTINCT

- Composició o Combinació JOIN
  - R3 = R1 ⋈C R2
    R3 nova relació i R1 i R2 són relacions
    C és una condició sobre components de R1 i R2
    R3 conté producte cartesià de R1 i R2 q compleixen la condició C

  Variants JOIN:
  - OUTER JOIN: conserva tuples que no entren al join completant-les amb NULLS
    - LEFT JOIN: totes tuples taula esquerra
    - RIGHT JOIN: totes tuples taula dreta
    - FULL JOIN: totes tuples ambdues taules

  - EQUI-JOIN: condició son igualtats entre dos relacions
    SI tenen el mateix les dues taules, clàusula USING indica omet una de les components repetides (no tots sistemes accepten clàusula)
  - NATURAL JOIN: és un EQUI-JOIN on aplica igualtat a totes les components dels només que coincideixin (intentar no fer servir)

.

- Agregats GROUP BY

Processos d'agrupament, agrupen per separat les tuples en funció del valor que tinguin indicat, es poden obtenir: recomptes, max, mins, totals mitjans ..
Cada tupla correspon al grup de tuples de la relació original

Materialització:
Relació materialitzada: es conserva a la memòria en forma de taula
Relació no materialitzada: respostes a consultes (poden ser reutilitzades subquery)
Vistes: taula però es defineixen com a resultat de consulta

- Reanomenament AS

Per relació existent, es manté temporalment fins completar la consulta
(duplicat d'una taula sense gastar recursos)

- Ordenació ORDER BY

Les tuples amb un ordre determinat per presentació o tractament
diferents millores temps:

- eliminació duplicats + DISTINCT
- JOINS temps sub quadràtic
- crear índex de tipus clustered

- Valor NULL

component o atribut buit, limitar sempre el seu ús
Dependrà de tipus de unitats si accepta igualacions, per defecte totes columnes accepten NULL- menys si es defineix com NOT NULL

Laboratori:

```
scp -r userubiwan@ubiwan.epsevg.upc.edu:/home/públic/dabd/02accounts/
/home/mariona/Documentos/altres-DABD/

taula bàsica de contactes:

sqlite> CREATE TABLE contactes (
   ...> id INTEGER,
   ...> firstname TEXT,
   ...> lastname TEXT NOT NULL,
   ...> email TEXT NOT NULL UNIQUE,
   ...> PRIMARY KEY(id)
   ...> );
sqlite> .tables
contactes

sqlite> INSERT INTO contactes VALUES(1,'ola','gonazalez','aa@gmail.com');
sqlite> SELECT* FROM contactes;
1|ola|gonazalez|aa@gmail.com

borrar una taula:
sqlite> DROP TABLE accounts;
```

Crear taula accounts:
```
sqlite> CREATE TABLE accounts (
   ...> acc_id INTEGER,
   ...> type CHAR(1),
   ...> balance REAL,
   ...> owner TEXT,
   ...> owner_id INTEGER,
   ...> phone INTEGER,
   ...> address TEXT)
   ...> ;
```

importar valors del .txt a la taula:
```
sqlite> .import accounts.txt accounts
```

veure si s'han inserit correctament:
```
sqlite> SELECT * FROM accounts;
acc_id|type|balance|owner|owner_id|phone|address
161320011440|P|1763.68|Tomé Cecial|6463525|223242|Camino de Criptana 2,
Argamasilla
161320011440|P|1763.68|Sancho Panza|6532345|222333|Corrales s/n, Argamasilla
111930116980|C|1564.27|Álvaro Tarfe|6112452|334455|Avellaneda 2, Granada
```

crear taules +inserir des de .sql:
```
 sqlite3 accounts2.db < accounts.sql
```

Consultes:
bd: accounts
types : acc_id|type|balance|owner|owner_id|phone|address

tots dnis i únic:
```
sqlite> SELECT DISTINCT owner_id FROM accounts;
```
dni persona en particular:
```
sqlite> SELECT owner_id FROM accounts WHERE owner
='Pedro Alonso';
6564321
6564321
```

comptes saldo superior 1000:
```
sqlite> SELECT owner FROM accounts WHERE balance
<1000;
Caballero de los Espejos
Pedro Alonso
Roque Guinart
```

Comptes q no son tipus L:
```
sqlite> SELECT  acc_id,owner,type FROM accounts WHERE
type !='L';
161320011440|Tomé Cecial|P
161320011440|Sancho Panza|P
111930116980|Álvaro Tarfe|C
134300219657|Sancho de Azpetia|C
```

Saldo disponible per cadascun titulars (sumant comptes)sense repeticions:
```
sqlite>
SELECT owner,balance, SUM(balance) AS total_balance FROM accounts GROUP BY
owner_id;
Ginés de Pasamonte|1551.99|11624.81
Vivaldo Cachopín|1243.68|1243.68
Sancho de Azpetia|1438.91|4139.06
```

nom i telf sense repeticions dni del alonso quijano (subquerys): sqlite> SELECT DISTINCT owner, phone FROM accounts WHERE owner_id = (SELECT owner_id FROM accounts WHERE owner = 'Alonso Quijano' LIMIT 1);
Alonso Quijano|213243
Caballero de la Triste Figura|213243

parells noms mateixa persona,dni, sense repeticions(joins 5tuples):

Compte saldo major: sqlite> SELECT DISTINCT owner, phone FROM accounts WHERE owner_id = (SELECT owner_id FROM accounts WHERE owner = 'Alonso Quijano' LIMIT 1);
Alonso Quijano|213243
Caballero de la Triste Figura|213243

Compte saldo menor: sqlite> SELECT * FROM accounts WHERE balance > 0 ORDER BY balance ASC LIMIT 1;
171174310952|C|28.89|Caballero de la Blanca Luna|6435323|234678|Mayor 11, Argamasilla

Tema 2: Modelat Conceptual
1/80
Models: representacions simbòliques de la realitat
Conceptuals: basat en conceptes -> esforç d'abstracció

Dificultats:
- incoherencia: solució no és única
- saltar fets rellevants del context
- Grau adequat fidelitat representació:
    - nivell detall insuficient i excessiu és dolent
- Nivell adequat detall varia segons objectiu: d'usuaris, desenvolupadors, client..

Imprescindible saber QUÉ estem modelant:
- Quines dades de la realitat anem a processar?
- Qui les vol processar? Per què?
- Quins graus d'eficàcia i detall volem?

És crucial:
que el model suposi un progrès en la nostra comprensió de la realitat a modelar;
Per aconseguir-ho, el model aporta simplicitat.

Opcions modelat:
- Esquemes relacionals:
  A partir de les SQL, podem traduir directament per crear una bd
- Diagrames E/R:
  (Entitat / Relació)
- Diagrames de classe UML:
  Especificació classes un entorn OOP, tenen ajustos notacionalsi doble caixa per representar una entitat dèbil

Esquemes relacionals més propers nivell de la propia bd, altres estan més aprop nivell del humà de raonar i modelar

→ Centrarem en diagrames UML deduits per esquemes relacionals

Elements de modelat:
Taules i claus en models relacionals, entitats i relacions en E/R, classes i en UML...
- Cada element d'un model ha de ser la contrapartida clara d'un fet o percepció de la realitat que estem modelant
- ha de ser un fet o percepció rellevant
- tots els fets o percepcions rellevants han d'estar recollits en el model
- han d'estar-hi un sol cop (DRY: Don't Repeat Yourself)

Criteris:
- Cada element tenir un significat clarament diferent del significat de la resta (DRY).
- Un model gran és probablement millorable si s'estructura millor.
- Un model complicat és probablement millorable si s'analitza millor.
- El significat de cada element del model es pugui explicar a un usuari no informàtic
- Els casos en que un element pot no correspondre a res (per ex. atributs NULL) han de ser molt excepcionals.

Són necessaris esquemes relacionals.
Para cada relació de la Base de Dades:

- nom
- noms dels atributs
- documentació addicional (connexions entre les relacions i altres restriccions d'integritat).

Dues maneres d'arribar-hi:

- diagrames E/R o diagrames de classes
- la teoria de la normalització (anàlisis de les dependències funcionals).

Hem de ser capaços d'aplicar ambdós.

Intermedis abstracció:

- classes
- atributs
- mètodes
- associacions entre classes
- multiplicitats
- classes associatives
- especialitzacions (triar en subtipus dins d'una mateixa classe-herències)

Es verbs corresponen a mètodes
(verbs "te" o "consta que" es poden referir atributs respecte classes o associacions classes)
noms comuns a classes o atributs

- **Multiplicitats:**

Quants objectes de classe llunyana es relaciona amb classe propera

- 0..1 (0 o 1)
- 1 (si o si 1)
- 0..* (min 0 a *)
- 1..* (min 1 a *)

Classe o atribut?
El mantenim com atribut si la seva estructura interna és irrellevant (indivisibilitat).
Passa a ser una classe si:

- cada valor de l'atribut consta òbviament de vàries parts rellevants
- potser ens manca el valor (recordem la discussió sobre NULL)
- ens agradaria que tingués varis valors simultàniament
- trobem difícil concretar el domini
- el mateix nom d'atribut apareix vàries vegades i amb la mateixa semàntica al llarg del procés de modelat…

Dificultat crucial: Posar-hi suficient imaginació, però no massa!

llista de coses → Taula (seleccionar dins de la llista fixa ex: països)

bd també té: Relacions i les seves restriccions

L'esquema de relació:
(Com create table) consta:

- nom de la relació,
- nom i domini de cadascun dels atributs, amb les seves

possibles condicions NOT NULL,
claus:

- clau primària (PRIMARY KEY) → subratllades
- claus alternatives (UNIQUE)
- claus foranes (REFERENCES) i el seu ús.

SQLite .schema

- **Claus:**

conjunt de un o més atributs que:

- determinen tota la relació
- és petits possibles
- una clau s'escull com a primària, altres son alternatives

Claus primàries o alternative NO poden ser nuls
Un o més atributs d'una relació´ es pot "referir" a una altre relació guardant un valor de la seva clau primària (clau forana).
→ Els atributs de la clau forana apunten a una tupla existent en la taula referida ("restricció d'integritat referencial").

Clau forana ha de tenir mateixos atributs que la clau primària que apunta

Què fer quan la clau primària externa referida canvia el seu valor?
→ Clàusula de propagació "ON UPDATE".
Què fer quan s'elimina la tupla identificada per la clau primària externa referida?
→ Clàusula de propagació "ON DELETE".

SQLIte claus foranes operatives : **PRAGMA foreign_keys=ON;**

Accions claus foranes:

- NO ACTION
- RESTRICT - si hi han extra referències no borrar
- CASCADE - borrar amb totes les referències
- SET NULL
- SET DEFAULT

46/80
És convenient si apareixen multiplicitat 1 o 0..1 implementar associacions foranes
Així te dependencia funcional

- **Especialitzacions:**

Representa assegurant q les tuples d'una classe especialitzada mantinguin com a clau primària la mateixa que s'usa en classe general -- relaciona diferents classes fent especificacions

Claus dèbils: classes que la seva clau primària conté una clau forana → son dependents a altres claus

- Doble caixa en el diagrama de classes

- **Anomalies**

Son situacions que en el disseny de les bases de dades ens dificulta mantenir informació correctament.
Exemple:

Funcionari (NRP, Nom, Destí, Estat, DataEstat)
- *Inserció de Nom i NRP si encara no sap el Destí?*
- *Eliminació si elimines la info dels Destins del funcionari, perds la relació NRP i Nom*
- *Actualització si canvia Nom a un NRP, sha d'actualitzar tots els valors*

Problema NRP DETERMINA NOM
- *existeix retundància de la informació*

Dependència funcional:
Existeix una dependència funcional P → Q (llegit: "P determina Q")
si succeeix el següent:
Sempre que dos tuples tinguin els mateixos valors en els atributs de P, forçosament hauran de tenir també els mateixos valors en els atributs de Q.

Dependències funcionals mal gestionades implica esquemes relacionals mal dissenyats:
- dades redundants i usant clàusules DISTINCT en SELECT (consultes lentes)
  SELECT DISTINCT NRP, Nom FROM Funcionari;
- dificulten els joins -> lossy-join perden informació

Valor d'alguns atributs poden determinar valor d'altres atributs:
- El Número de Registre de Personal determina el nom del funcionari.
- Els atributs de direcció postal (ciutat, carrer i número) determinen conjuntament el codi postal.
  → Conjuntament pq en separat no ens donara el c postal correcte
- Habitualment s'espera que el número de DNI determini totes les dades de la persona.
- En una aplicació real no te'n pots fiar del tot.
- Una clau primària determina tots els valors de la tupla.

Una factoria immensa té una elevada quantitat de màquines. De cadascuna tenim el número amb que està inventariada i el seu fabricant.
Periòdicament, un tècnic supervisa individualment cada màquina.
Cada tècnic, de qui coneixem el DNI i les seves dades personals, coneix la distribució de la seva setmana laboral en franges horàries amb la indicació de quina màquina ha de supervisar en cadascuna d'aquestes franges.

dependències:
Ninventari - Ninv
Fabricant- Fab
Dni
DadesPersonals - DP
franjahoraria- FH

Ninv -> Fab
Dni -> DP
Dni + FH -> Ninv
Ninv + FH -> Dni (tecnic supervisa individualment una maquina a una fh)

Formes normals:
Existeixen dependencies inevitables -> CLAUS causen dependències funcionals
Per evitar anomalies ->NORMALIZAR BD majoria dependencies es converteix en claus de les relacions

FORMES NORMALS (NF)
Tenim relació R no està normalitzada, sha de descomposar en varies relacions R1,R2 ..Rn

- Cada nova relació Ri conté un subconjunt d'atributs de R
- Cada atribut de R apareix com un atribut en almenys una de les noves relacions Ri

Cal assegurar-nos que:

- Descomposició que sigui lossless-join (sempre - Sense pèrdues d'informació)
    - La descomposició de la relació R en relacions més petites R1 , R2 . . . Rn és lossless-join si:
    - Tots els atributs de R apareixen en la descomposició:

  Atributs(R) = Atributs(R1) ∪ Atributs(R2) ∪ . . . ∪ Atributs(Rn)
    - Som capaços de reconstruir R fent natural joins de les relacions petites:

  R = R1 ./ R2 ./ . . . ./ Rn
- No hi hagi redundàncies (BCNF, 3NF no sempre totes)
- Preservació de DF (3NF, BCNF no sempre totes)

1NF 1ra Forma Normal:
Cada valor de cada columna sigui atòmica (no múltiples valors) ni atributs repetits (ex: Assig1 ,Assig2, Assig3…)

Exemple:

CodiEst	NomEst	Ciutat	Comarca	Assig	DataAprovat
E101	Marta	Sitges	Garraf	DABD	01-07-2019
E101	Marta	Sitges	Garraf	PMUD	30-01-2020
E102	Pol	Piera	Anoia	DABD,AMEP	28-06-2019
E103	Berta	Vilanova	Garraf	ESIN	01-02-2020
E104	Pau	Sitges	Garraf	DABD	01-07-2019

Ha de quedar:

CodiEst	NomEst	Ciutat	Comarca	Assig	DataAprovat
E101	Marta	Sitges	Garraf	DABD	01-07-2019
E101	Marta	Sitges	Garraf	PMUD	30-01-2020
E102	Pol	Piera	Anoia	DABD	28-06-2019
E102	Pol	Piera	Anoia	AMEP	28-06-2019
E103	Berta	Vilanova	Garraf	ESIN	01-02-2020
E104	Pau	Sitges	Garraf	DABD	01-07-2019

(dividir tots els que tinguin múltiples valors)

2NF 2na Forma Normal:
Cal que la relació sigui 1NF i qualsevol ATRIBUT NO CLAU depengui de la clau sencera

CodiEst	NomEst	Ciutat	Comarca
E101	Marta	Sitges	Garraf
E102	Pol	Piera	Anoia
E103	Berta	Vilanova	Garraf
E104	Pau	Sitges	Garraf

CodiEst	Assig	DataAprovat
E101	DABD	01-07-2019
E101	PMUD	30-01-2020
E102	DABD	28-06-2019
E102	AMEP	28-06-2019
E103	ESIN	01-02-2020
E104	DABD	01-07-2019

(Codi Est i Assaig son les claus i atributs: NomEst, Ciutat i Comarca no dependen de res→ posar que depenguin de CodiEst per satisfer 2nf)

3NF 3ra Forma Normal:
Cal que la relació sigui 2NF i que no tingui dependències transitivess dins seu

CodiEst	NomEst	Ciutat
E101	Marta	Sitges
E102	Pol	Piera
E103	Berta	Vilanova
E104	Pau	Sitges

Ciutat	Comarca
Sitges	Garraf
Piera	Anoia
Vilanova	Garraf

La relació dels estudiants no Comarca depèn de Ciutat i Ciutat depèn de codiEst-> cal partirlo en 2 per satisfer 3nf

Formes normals:

- tercera forma normal **3NF** la més usada
- forma normal boyce-Codd **BCNF** tmb anomenada 3.5NF

diferents en grau eviten dependències evitables:

- En BCNF no existeixen dependències funcionals, excepte les inevitables (degudes a les claus);
- 3NF es permeten algunes que són evitables, si quan les evitem perdem informació que volem conservar sobre altres dependències.

$\mathcal{R}$ està en forma normal de Boyce-Codd (BCNF) si:
sempre que $X \to A$, per $X \subseteq \mathcal{R}$ i $A \in \mathcal{R}$,
$A \in X$ o bé X conté una clau de $\mathcal{R}$;

$\mathcal{R}$ està en tercera forma normal (3NF) si:
sempre que $X \to A$, per $X \subseteq \mathcal{R}$ i $A \in \mathcal{R}$,
$A \in X$ o bé X conté una clau de $\mathcal{R}$,
o bé A pertany a una clau (primària o alternativa).

Igualment s'aconsegueix normalitzar els esquemes - descomposar la relació en relacions més petites en que les dependències corresponguin a claus

Ninv -> Fab
Dni -> DP
Dni + FH -> Ninv
Ninv + FH -> Dn
⇒ *Maq (Niv, Fab)*
⇒ *Tec (Dni, DP)*
⇒ *Supervisió(Ninv, Dni, FH)*
Intersecció:
Maq (Niv, Fab) / Supervisió(Ninv, Dni, FH) → Ninv (que determina Fab)
⇒ *Tec (Dni, DP) /Supervisió (Ninv, Dni, FH) → Dni (determina Dp)*
Per exemple:
R = {A, B, C , D, E }
DF = {A → BC , B → D, CD → E }
I R1 = {A, B, C} R2 = {A, D, E} és una descomposició lossless-join?
Intersecció A determina BC (R1)
I R1 = {A, B, C} R2 = {C, D, E} és una descomposició lossless-join?
Intersecció C que NO determina res - MALA descomposició

40-46/80

Laboratori:

EXTREURE DADES SQLITE:
treure schema
des de terminal : sqlite3 aprovats.db .schema> aprovats1.sql
ARA entrant a SQLITE3 aprovats.db
sqlite3 aprovats.db
Copiar MANUALMENT tots els inserts de les diferents taules:
. mode insert aprovat
SELECT * FROM aprovat;
INSERT INTO aprovat VALUES('C1813','Gabriel Pla Madrigal','Sant Pere de Ribes','Garraf',4.2999999999999998223,'2018-01-07','ARCO','Q4');
INSERT INTO aprovat VALUES('C4864','Alicia Losada Sánchez','Sant Pere de Ribes','Garraf',6.4000000000000003552,'2018-06-21','PRO1','Q2');
..
sqlite> .mode insert ASSIGNATURA
sqlite> SELECT * from AS
AS ASC ASSIGNATURA
sqlite> SELECT * from ASSIGNATURA ;
INSERT INTO ASSIGNATURA VALUES('ARCO','Q4');
INSERT INTO ASSIGNATURA VALUES('PRO1','Q2');
..

ENTRAR SSH:
conectar ssh per entrar a les bds:
ssh userubiwan@ubiwan.epsevg.upc.edu
userubiwan@ubiwan.epsevg.upc.edu's password: xxx

ENTRAR MYSQL:
mysql -u est_userubiwan -p
Enter password: dB.userubiwan

entrar a la base de dades de l'usuari:
mysql> \u est_userubiwan

OBJ

IMPORTAR FITXERS A MYSQL: accounts i movies
dins de debian!
userubiwan@ubiwan:~$ mysql -u est_userubiwan -p est_userubiwan < accounts.sql
Enter password:

(ara accounts estara a mysql)

EXPORTAR MYSQL:
dins de debian!
userubiwan@ubiwan:~$ mysqldump -u est_userubiwan -p est_userubiwan pets --no-tablespaces --compatible=ansi > datapets.sql
Enter password:

o de la següent manera:
userubiwan@ubiwan:~$ mysqldump -u est_userubiwan -p est_userubiwan pets > datapets.sql --column-statistics=0 --no-tablespaces

Enter password: dB.userubiwan

userubiwan@ubiwan:~$ ls
DABD PROP accounts.sql datamovies.sql elasticsearch-logs
INTE REIN assignatures datapets.sql fotos
MIDA XAMU datamovie.sql elasticsearch-data mnopublic.sql

ENTRAR POSTGRESQL:
dins de debian!
userubiwan@ubiwan:~$ psql -h ubiwan.epsevg.upc.edu -U est_userubiwan -W
Password: dB.userubiwan
psql (12.18 (Ubuntu 12.18-0ubuntu0.20.04.1))
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256, compression: off)
Type "help" for help.

(ja dins de la database)

IMPORTAR FITXERS A POSTGRESQL: pets i accounts
dins de debian!
userubiwan@ubiwan:~$ psql -h ubiwan.epsevg.upc.edu -U est_userubiwan est_userubiwan -f accounts.sql
Password for user est_userubiwan: dB.userubiwan
CREATE TABLE
INSERT 0 1
INSERT 0 1
…

userubiwan@ubiwan:~$ psql -h ubiwan.epsevg.upc.edu -U est_userubiwan est_userubiwan -f datapets.sql
Password for user est_userubiwan:

EXPORTAR POSTGRESQL:
dins de debian!
userubiwan@ubiwan:~$ pg_dump -h ubiwan.epsevg.upc.edu -U est_userubiwan est_userubiwan -t movies --no-tablespaces --no-owner --no-acl --column-inserts > datamovies.sql
Password: dB.userubiwan

POSSIBLES ERROR COMPTABILITAT
MANUALMENT FER:
comentar tots els sets! :

```
SET statement_timeout = 0;
SET lock_timeout = 0;
SET idle_in_transaction_session_timeout = 0;
..
SET client_min_messages = warning;
SET row_security = off;

SET default_table_access_method = heap;
```

TREURE TOTS ELS PUBLIC. DEL CREATE I INSERTS:

```
CREATE TABLE public.movies (
    name text,
    year integer,
    director text,
    score integer
);
INSERT INTO public.movies (name, year, director, score) VALUES ('the
shining', 1980, 'Stanley Kubrick', 8);
```

Si es treu si es pot importar a mysql:

userubiwan@ubiwan:~$ mysql -h ubiwan.epsevg.upc.edu -u est_userubiwan -p est_userubiwan < datamovie.sql

Enter password:

INFORMATION SCHEMA:
Informació del schema de les bases de dades creades:

MYSQL:
Sortir de la meva base de dades
entrar a information_shemamysql> \u information_schema

```
mysql> SELECT table_name, column_name, column_type  FROM columns WHERE
table_name='accounts';
+------------+-------------+--------------+
| TABLE_NAME | COLUMN_NAME | COLUMN_TYPE  |
+------------+-------------+--------------+
| accounts   | acc_id      | bigint       |
| accounts   | type        | char(1)      |
| accounts   | balance     | double       |
| accounts   | owner       | varchar(40)  |
| accounts   | owner_id    | int          |
| accounts   | phone       | int          |
| accounts   | address     | varchar(100) |
+------------+-------------+--------------+
7 rows in set (0.00 sec)
```

POSTGRE SQL:

```
est_userubiwan=> \dnS
       List of schemas
       Name       |  Owner
--------------------+----------
 information_schema  | postgres
 pg_catalog          | postgres
 pg_temp_1           | postgres
 pg_toast            | postgres
 pg_toast_temp_1     | postgres
 public              | postgres
(6 rows)
```

```
est_userubiwan=> SELECT udt_catalog, column_name, udt_name FROM
information_schema.columns WHERE table_name='accounts';
 udt_catalog  | column_name | udt_name
--------------+-------------+----------
 est_userubiwan | acc_id      | int8
 est_userubiwan | type        | bpchar
 est_userubiwan | balance     | float4
 est_userubiwan | owner       | varchar
 est_userubiwan | owner_id    | int4
 est_userubiwan | phone       | int4
 est_userubiwan | address     | varchar
(7 rows)
```

Laboratori:

Sessio 5: de php i html
inici web: https://ubiwan.epsevg.upc.edu/~userubiwan/
configuració de base de dades .php:

```
// Host, nom del servidor o IP del servidor MySQL.
$sql_host = "localhost";

// Usuari/contrasenya de MySQL i nom de la base de dades
$sql_user = "est_userubiwan";
$sql_passwd = "dB.userubiwan";
$sql_db = "est_userubiwan";
```

mirar consulta : https://ubiwan.epsevg.upc.edu/~userubiwan/users.html

Connectat al servidor MySQL...

Nom: mariona
Contrasenya: anoriam

Connexió tancada

mirar llistat users: https://ubiwan.epsevg.upc.edu/~userubiwan/list_users.php

Connectat al servidor MySQL...
Nom: mariona
Contrasenya: anoriam

Nom: donald
Contrasenya: quack

Nom: a
Contrasenya: aa

Nom:
Contrasenya:

Connexió tancada

mirar add users: https://ubiwan.epsevg.upc.edu/~userubiwan/add_users.html

Per gestionar una taula de noms i contrasenyes:

Llista usuaris...
Nom: prova2
Contrasenya: contraprova2
Envia...

ara en llistat surt:

Connectat al servidor MySQL...
Nom: mariona
Contrasenya: anoriam

Nom: donald
Contrasenya: quack

Nom: a
Contrasenya: aa

Nom:
Contrasenya:

Nom: prova add
Contrasenya: provi

Nom: prova add
Contrasenya: provi

Nom: prova2
Contrasenya: contraprova2

Connexió tancada

Tema 3: índexs, Transaccions Disparadors

Accelerar determinades consultes:
- crear index
- alentir altre operacions

Índexs:
Els índexs es creen per accelerar consultes específiques, seleccionades curosament. Aquesta millora en el rendiment ve amb el cost d'invertir temps en la creació de l'índex i potencialment alentir altres operacions.

Es descriuen dos estils principals d'estructures d'índexs: variants de hashing (com ara hash dinàmic lineal i hash dinàmic extensible) i variants d'arbres de cerca equilibrats (predominantment l'arbre B+).
L'ús exclusiu dels arbres B+ en bases de dades relacionals es justifica pel seu rendiment superior en la gestió d'índexs. Els arbres B+ faciliten la cerca ràpida a través de claus indexades que apunten a les ubicacions de les dades.

Escollir be quines consultes es necessiten per un índex

Estructura sobre la cerca que permet trobar files a partir de valors dels seus camps:
- Util per seleccions amb criteri d'igualtat (WHERE a =...)
- Útil per restriccions UNIQUE o PRIMARY KEY.

Dos estils en estructures tradicionals cerca eficient:
- Variants de hashing:
    - hash dinàmic lineal
    - hash dinàmic extensible
- Variants d'arbres equilibrats de cerca: el més comú és arbre B+

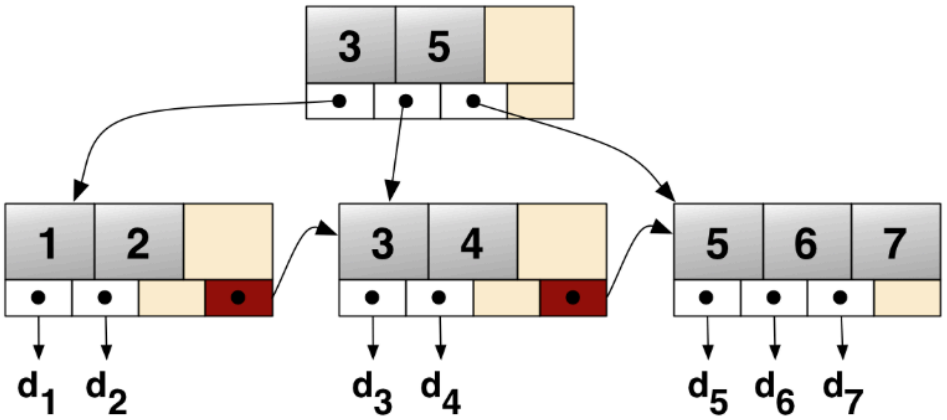
Arbres B+:
- Arbres binaris de cerca o BST
- Arbres binaris equilibrats o AVL
- Arbres 2-3, arbres red-black (que son equivalents a els arbres 2-3-4)
- arbres AA
- Arbres B i B+

DB només s'usen els arbres B+

- BST(Binary Search Trees): Arbres binaris de cerca: Classifiquem els elements segons la clau per no haver-los de recorrer tots quan cerquem.
- AVL: BST equilibrats en altura. Evitem la degeneració dels BST reequilibrant-los quan fa falta.
- Arbres 2-3: cada node conté una o dos claus:
    Els nodes que contenen una clau tenen dos subarbres i s'usa el mateix criteri que en els BST.
    Els nodes que contenen dos claus tenen tres subarbres i s'usa una extensió del criteri BST:

    - Igual que BST respecte al subarbre esquerra i el dret, comparats amb les dues claus.
    - El subarbre central conte claus estrictament entre les dos claus del node.
- Arbre 2-3-4: cada node conte una, dues o tres claus, tenint dos, tres o quatre subarbres respectivament.
- Arbre red-black: Estructuralment equivalent a l'arbre 2-3-4, on cada node de l'arbre 2-3-4 correspon a un node negre que te zero, un o dos nodes fills vermells.
- Arbre AA: Variant de l'arbre red-black en que els nodes vermells només poden ser afegits com a fills drets, simplificant els 7 diferents casos de balanceig dels red-black a només 2 dels AA.

Arbre B i B+:
B de balanced - cada node (excepte l'arrel) conte entre d i 2d claus i entre d + 1 i 2d + 1 subarbres. Per d = 1 son els arbres 2-3.
En un arbre B+, les claus dels nodes interiors es repeteixen a les fulles, de manera que recorren les fulles es recorren totes les claus.
Les fulles formen part d'una llista doblement enllaçada per poder-les recorrer ASC i DESC



L'ındex que usa un arbre B+ ha de fer tres passos:
1. Recórrer els nivells de l'arbre B+ des de l'arrel fins a una fulla
2. Recorrer la llista doblement encadenada de les fulles del B+
3. Saltar de les fulles del B+ a les files de la taula
Si les claus no estan repetides i fem cerques d'igualtat, no cal fer el 2 on pas.
Si la informació que cerquem està a la fulla (per exemple veure si la clau existeix o no) o es un índex clustered (les files de la taula ja estan en les fulles), no cal fer el 3er pas.

Podem usar un o varis atributs de la relació com clau de cerca en
l'ındex (arbre B+). Però, llavors,
A on estan les tuples?
- Index clustered (o a vegades primari): Les tuples es troben a les fulles de l'arbre B+.
- Index unclustered (o a vegades secundari): Les tuples estan en "un altre lloc" i les fulles de l'arbre B+ contenen punters a les corresponents tuples.

Si hi ha un ındex clustered, forçosament les tuples estan en memoria fısicament ordenades d'acord amb la clau de cerca de l'arbre B+ clustered.
Per tant, una mateixa taula no pot tenir més que un ındex clustered.

CLAUS:

És important distingir que estem fent varis usos d'aquest terme amb significats diferents:

- Clau primària d'una relació (PRIMARY KEY): s'escull per identificar cada tupla.
- Clau forana (FOREIGN KEY): clau primària d'una altra taula, permet relacionar una taula amb una altra.
- Clau alternativa d'una relació (UNIQUE): es podria escollir per identificar cada tupla en lloc de la clau primària.
- Clau de cerca d'un índex sobre una relació: atributs els valors dels quals ens permeten accelerar la recuperació de les tuples que volem sense recórrer tota la taula.

Tota clau declarada adquireix automàticament un ındex,

- tant claus primàries, PRIMARY KEY,
- com claus alternatives, UNIQUE;

Motiu principal: poder comprovar ràpidament la absència de duplicats en les insercions.
Aquest ús, per tant, correspon a condicions d'igualtat.
Les operacions d'igualtat no es beneficien particularment de tenir un ındex clustered.
També hi hauran els ındex que creem manualment:

CREATE INDEX nom index ON taula(atributs)

Podem usar una cláusula WHERE quan creem l'ındex. Només les tuples que compleix la condicio son indexades.

CREATE INDEX nom index ON taula(atributs) WHERE condicio

Es útil per accelerar consultes on el WHERE usa condicions amb valors constants. Aixı pot usar l'ındex parcial que és més ràpid al ser més petit.

Un covering index es aquell que conté tots, i potser més, els atributs necessaris per la query.
Per exemple, si fem aquesta query:

SELECT column1, column2 FROM tablename WHERE criteria

i tenim un ındex que pot accelerar aquesta consulta i a mes conté els atributs column1, column2, no caldria anar a les tuples per recuperar la informació, sinó que podem obtenir els resultats directament de l'ındex.
Si aquesta és una consulta habitual que cal acelerar, és convenient crear un índex que almenys contingui els atributs column1, column2.

Quan NO crear índexs:

- Taules petites.
- Taules que sovint tenen grans operacions batch d'UPDATE o INSERT.
- Columnes que contenen un gran nombre de valors NULL.
- Columnes que son manipulades freqüentment.
- Quan ja tenim un altre ındex que inclou les mateixes columnes al principi (ındex B+).

Per ex. si tenim taula persones amb un ındex format per les columnes cognom i nom, no cal fer un ındex per cercar només per cognom (però si que cal si volem cercar només per nom).

Transaccions
22/77
Una transacció és una seqüència d'operacions i consultes que formen una única operació conceptual des del punt de vista humà. Les transaccions asseguren que les bases de dades mantinguin la coherència i l'estat íntegre, fins i tot quan s'executen operacions simultànies de múltiples clients.

S'aborden els problemes de concurrència quan múltiples clients interactuen amb la mateixa base de dades. Conceptes com a bloquejos, actualitzacions perdudes, i lectures brutes, enfocant en com les transaccions ajuden a manejar aquests reptes.

Coherencia:
La base de dades és un reflex fidel de la realitat.En cap cas es pot responsabilitzar de que la transacció estigui mal programada;
però si que pot garantir l'execució sencera de la transacció:
estalvia al programador el preocupar-se de si la seva transacció s'ha reflectit correctament a la base de dades;
i també pot garantir l'execució ıntegra i sense interferències de la transacció:
permet al programador programar sense saber que altres transaccions estaran rondant al voltant;
lo que correspon a les propietats de
atomicitat: la transacció, si s'executa, ho fa en la seva totalitat, i
aïllament: l'existència d'altres transaccions concurrentment no afecta a la transacció

ACID:
Atomicity, atomicitat; Cada transacció es realitza completament, o bé no es realitza en absolut i el SGBD ens avisa
Consistency, coherència; Així, si les transaccions estan ben programades,
Isolation, aïllament; garanteix que actuarà com si cada transacció fos l'única transacció activa:
Durability, durabilitat;Les modificacions realitzades es conservaran
El SGBD ofereix a les transaccions dos opcions per acabar:
COMMIT: la transacció es compromet a fer-ho tot bé.
ROLLBACK: si us plau, que es desfaci tot el que la transacció ha fet i la base de dades quedi com abans de que comencés la transacció.
Nivells estàndards d'aïllament:
I SERIAL,
I SERIALIZABLE,
I REPEATABLE READ,
I READ COMMITTED,
I READ UNCOMMITTED.
Garanties absolutes: Únicament SERIAL i SERIALIZABLE.
Els altres nivells obren la porta a interferències.

SERIAL:
Mai comença una transacció abans d'haver acabat l'anterior. Garanteix l'absència d'interaccions entre transaccions. Lent → I Amb freqüència, inadmissiblement lent

SERIALIZABLE:
SGBD pot intercalar transaccions com li sembli oportú, però el resultat net ha de ser el mateix que el d'una execució serial. Garanteix l'absència d'interaccions entre transaccions → Molt més eficient que l'execució o serial - pot resultar ineficient.

Interferències:

- **Lost Update** (Actualització perduda): una transacció modifica una dada que una altra transacció creu conèixer perquè s'acaba d'escriure.
- **Uncommitted Read o Dirty Read** (Lectura no confirmada): una transacció llegeix dades modificades per una altra transacció que encara no ha acabat.
- **Unrepeatable Read (**Lectura no repetible): una transacció modifica una dada que una altra transacció creu conèixer perquè l'acaba de llegir.
- **Phantom Read**: Aparició de "fantasmes" al treballar amb conjunts de dades.

REPEATABLE READ:
Aquest nivell assegura que si una transacció llegeix una fila, cap altra transacció no podrà escriure aquesta fila fins que la primera transacció finalitzi. Això prevé les lectures no repetibles. Tanmateix, aquest nivell no necessàriament impedeix el que es coneix com a "lectura fantasma", on una transacció pot veure un conjunt de dades canviar si altres transaccions afegeixen o eliminen files que compleixen amb les condicions de cerca de la transacció inicial.

READ UNCOMMITTED:
Aquest és el nivell més baix d'aïllament. Permet a una transacció llegir dades que una altra transacció ha modificat però encara no ha confirmat (commit). Això pot conduir a les anomenades "dirty reads", on les dades llegides poden no ser consistents o fins i tot desaparèixer si la transacció que les va modificar es desfà.

READ COMMITTED:
Aquest nivell només permet a una transacció llegir dades que han estat confirmades per altres transaccions, eliminant així el problema de les lectures brutes. Tanmateix, si una transacció llegiu les mateixes dades múltiples vegades dins d'una única transacció, els valors llegits poden canviar si altres transaccions modifiquen aquestes dades i fan commit entre lectures successives. Això es coneix com a "lectura no repetible" o "non-repeatable read".

Nivell d'aïllament	Actualització perduda Lost Update	Lectura no confirmada Dirty Read	Lectura no repetible Nonrepeatable Read	Fantasma Phantom Read
`READ UNCOMMITTED`	possible	possible	possible	possible
`READ COMMITTED`	no	no	possible	possible
`REPEATABLE READ`	no	no	no	possible
`SERIALIZABLE`	no	no	no	no

Locks:
Associats a files, taules, pàgines,índex de lectura, o compartits o D'escriptura, o exclusius.
Si dos transaccions tenen panys associats al mateix objecte, han de ser ambdós compartits; cap dels dos pot ser exclusiu.
Una implementació dels nivells d'aïllament: el mètode de panys en dos fases (2PL, Two-Phase Locking).

REPEATABLE READ amb 2PL: Assegura que si una transacció llegeix dades, cap altra no pot modificar-les fins que finalitzi, prevenint així lectures no repetibles.
Les transaccions mantenir tots els panys de lectura fins al final de la transacció.

READ COMMITTED amb 2PL: Permet només llegir dades que han estat confirmades. Evita lectures brutes però no garanteix que les lectures siguin repetibles.
Les transaccions alliberen els panys de lectura immediatament després de la lectura i mantenir els d'escriptura fins al commit.

SERIALIZABLE amb 2PL: El més alt nivell d'aïllament que garanteix que les transaccions s'executen com si fossin seqüencials, prevenint tots els tipus d'interferències.
Utilitza panys extensius sobre taules o dades per evitar qualsevol accés concurrent que pugui conduir a interferències.

Disparadors
68/77
Els disparadors són procediments automàtics que s'executen en resposta a esdeveniments específics dins de la base de dades, com actualitzacions o insercions. Serveixen per a implementar regles de negoci, mantenir la integritat de la base de dades, i automatitzar processos de manteniment.

S'ofereix un exemple de com un disparador pot ser utilitzat per monitorar canvis en un compte bancari i registrar quan el saldo cau per sota de zero, demostrant la utilitat dels disparadors en situacions reals de negoci.

Usos habituals: I Implementació de restriccions addicionals que no es poden implementar amb claus. I Condicions dictades per la lògica de negoci de l'aplicació. I Detecció d'errors en temps d'execucióó, requerint un ROLLBACK de la transacció en curs i, possiblement, aixecant una excepció.

En clàusules ON DELETE i ON UPDATE:
El default en SQL estàndard és NO ACTION, però també s'admet declarar RESTRICT.

- RESTRICT correspon a comprovar si es va a violar la restricció d'integritat referencial (la clau forana ha d'adaptar a una tupla existent); si es va a violar, no es completa l'acció.
- NO ACTION indica que s'ha de realitzar l'acció i després comprovar si es viola la restricció d'integritat referencial (deferred check); si fos així es desfà el que s'ha fet.

Tema 4: Procés de consultes SQL

1. Descomposició de la consulta:
   I de quines operacions bàsiques consta i
   I en quin ordre s'hauran de realitzar (probablement hi ha varies possibilitats!).
2. Generació de plans, diverses maneres d'obtenir la resposta:
   I consideració de diversos algorismes per cada operación básica,
   I consideració de diversos camins d'accés (access path) a les tuples necessàries de cada taula,
   I reordenament (i possible modificació) de les operacions bàsiques que mantinguin el mateix resultat.
3. Optimització: avaluació predictiva del temps que requerirà cada possible pla i elecció del més prometedor;
4. Execució del pla seleccionat.

Fases:
1. "Tokenització": identificació de les seqüències de caràcters amb significat propi ("SELECT", >=, identificadors de taula o de columna. . .);
2. "Parsing" o anàlisis sintàctic: formules que van amb cada WHERE són correctes (operadors unaris, binaris). . .
3. "Análisis semàntic": resolució de les referències de cada identificador, comprovació de que els tipus en les operacions són correctes. . .

El resultat és un primer pla que, d'executar-se, proporciona la resposta correcta.
Però no necessàriament és la manera mes àgil d'obtenir-la.

SELECT:

Dos casos:

- El cas fàcil: només cal descartar els camps que no figuren en la projecció i conservar els que si figuren, en un simple recorregut de la taula.
- El cas laboriós: presència d'una cláusula DISTINCT.
- Comparar dos a dos totes les tuples resultat:
  - Cost quadràtic O(n2)
- Ordenar per evitar les repeticions:
  - Cost quasi lineal O(n log n) en general.
  - Però, si sabem que els repetits ja apareixen adjacents (per exemple, si hi ha un ORDER BY a més del DISTINCT, o si existeix un índex clustered), llavors només cal un recorregut.
    Cost lineal O(n)
- Existeixen altres solucions, algunes molt sofisticades.

Cas anàleg: GROUP BY, es recorre a ordenar, excepte si ja es té l'ordre degut a un índex clustered.

WHERE:

Es tracta d'aconseguir les tuples que compleixen la condició Terminologia habitual: access path a la taula, o, més informalment, com "atacar" la taula:

Recorregut de la taula, sempre disponible;
Ús d'un índex, si és que està disponible.

Preferible, si el resultat contindrà poques tuples (per exemple, si sabem que contindrà només una per involucrar en el WHERE una igualtat amb una PRIMARY KEY).

Desaconsellable si el resultat recuperarà moltes de les tuples de la taula: el overhead de l'ús de l'índex no es compensa.

En particular: mai posis un índex en una taula petita.

Covering index: quan existeix un índex que conté com a claus tots els atributs requerits, no s'arriba a mirar res de la taula.

JOIN:

Operació costosa però molt freqüent!

Per tant, moltes idees algorísmiques disponibles.

Nested loops: un bucle per relació, recorrent tuples; un dins de l'altre.

Es la corresponent selecció del producte cartesià, però sense materialitzar-lo.

Quina taula va al bucle intern i quina a l'extern?

Variants:

Block-oriented nested loops: un bucle per relació, recorrent pàgines de disc, un dins de l'altre; i, encara més a dins, un nested loops per recorrer tuples.

Index-based nested loops: el bucle interior es substitueix per un accés amb un índex, si aquest està disponible.

Join per sort-merge: s'ordenen ambdues relacions i s'aplica un algorisme de fusió. I Altres (per partició, hash join, hash join híbrid. . .)

Optimització de Consultes

L'optimitzador ha de prendre decisions sobre la combinació d'operacions, l'ordre d'aplicació, els camins d'accés a les dades, i com gestionar les relacions intermèdies. Es considera també l'ús d'índexs i altres estructures per millorar la velocitat i eficiència de les consultes.

Tema 5: Principis de l'administrador de BD
1/33

Tasques Principals de l'Administrador de Bases de Dades

- Gestió d'Usuaris i Privilegis: Responsabilitat d'assignar i controlar els accessos i permisos als usuaris de la base de dades.
- Gestió d'Incidències: Inclou la detecció, diagnòstic i correcció d'errors o problemes que poden sorgir dins de la base de dades.
- Optimització del Rendiment: Implica fer ajustaments tècnics per millorar la velocitat i eficiència de les operacions de la base de dades, com ara la selecció de quines operacions optimitzar sacrificant mínimament el rendiment en altres àrees.

Eines i Ajustaments Disponibles per a l'Administrador

- Organització dels Discos (Tablespaces): Optimitzar la configuració dels discos per millorar la velocitat i la redundància de les dades.
- Índexs: Creació o eliminació d'índexs per millorar la velocitat de consulta, amb un equilibri entre cost d'emmagatzematge i velocitat de recuperació.
- Nivells d'Aïllament de les Transaccions: Ajustar el nivell d'aïllament per equilibrar entre el rendiment i la coherència de les transaccions.
- Gestió de Pans/Locks: Optimitzar el maneig de pans per evitar bloquejos i permetre un alt grau de concurrència d'accés a les dades.

Cinc Principis Bàsics d'Administració

- Pensa Globalment; Actua Localment: Aquest principi general suggereix considerar l'impacte a llarg termini i àmpliament distribuït de les decisions locals.
- Accepta Negociar: Reconèixer que totes les decisions tècniques tenen compensacions i requerir una selecció cuidadosa entre diferents avantatges i desavantatges.
- Confia en l'Especialista: Delegar tasques específiques al sistema gestor de bases de dades, que està optimitzat per a realitzar certes operacions de manera eficient.
- Costa Més Començar que Seguir: Subratlla que iniciar processos o operacions pot ser costós, però una vegada establerts, els costos de manteniment són relativament baixos.
- Reparteix el Treball: Suggereix utilitzar tècniques de partició per distribuir la càrrega de treball, tant en termes d'espai (com en múltiples dispositius de disc) com de temps (aprofitant els períodes de baixa activitat).

Cada un d'aquests principis i conceptes són fonamentals per a l'administració efectiva d'una base de dades, proporcionant una base per a decisions estratègiques i tàctiques que maximizen el rendiment i l'estabilitat del sistema de bases de dades.

Tema 6: Bases de Dades NO SQL
1/23

Big data- conjunt de dades d'una mida a la frontera del que es manipulable amb la tecnologia disponible
Les 3 propietats del Big Data:

- Volum: Hi ha dades generades de moltes maneres
  - Per usuaris
  - Per empreses o organitzacions
  - Per logs
  - Per bots
- Varietat: Dades en qualsevol format: Text, CSV, full de càlcul, audio, vídeo, imatge, .
- Velocitat: Així que una data es guarda volem que ràpidament sigui accessible per altres

Límits del Model Relacional
Escalabilitat: El model relacional té limitacions quan es tracta d'escalar per manejar grans volums de dades distribuïdes.
  Scaling UP: preu creix i sostres de performance
  Scaling OUT: no sostres de performance, gestió i programació molt més complicada, dificultat en mantenir coherencia
Flexibilitat: Els esquemes rígids del model relacional no s'adapten bé als requisits canviants i a la naturalesa diversa de les dades modernes.

Teorema CAP:
Postula que és impossible per a un sistema distribuït garantir simultàniament la coherència, la disponibilitat i la tolerància a particions. Això implica compromisos en el disseny de sistemes distribuïts.

- **Coherència**: Després d'actualitzar un objecte, tot accés a l'objecte retorna el valor actualitzat (atomicitat de l'actualització).
- **Disponibilitat** (Availability): Tota consulta d'un programa client rep resposta.
- **Partició**: La base de dades està distribuïda en diversos servidors; és possible que es perdin missatges de comunicació entre ells (per exemple, un servidor pot caure).

No es poden assolir les tres propietats simultàniament.
Els sistemes NoSQL (no relacionals) admeten estar particionats i busquen equilibris útils entre:
  ”prou coherència” (potser no tota)
  ”prou disponibilitat” (potser no tota)

Demostració
No és gens difícil

Suposem les dades repartides en dos nodes, A i B:

- A rep una petició "read(x)";
- La coherència requereix que A esbrini si B ha atès un instant abans una petició "write(x,val)";
- A envia la pregunta a B però la xarxa perd el missatge.
- Una de dues:
  - A contesta sense esperar la resposta: pot ser incoherent si B ha canviat el valor de x, o
  - abans de contestar, A s'espera una resposta que mai arriba: es perd la disponibilitat.

Bases de Dades NoSQL

Característiques Principals:

- Descentralització: Distribució de dades a través de múltiples servidors o ubicacions geogràfiques.
- BASE (Basically Available, Soft state, Eventually consistent): Enfoc més flexible en termes de consistència de dades comparat amb el model ACID tradicional.
- Escalabilitat i Tolerància a Falles: Millor suport per a escalabilitat i alta disponibilitat.

- Inconvenients: Menys adequades per a transaccions complexes, OLAP i data warehousing degut a la manca d'operacions join complexes i altres funcionalitats SQL. Incompatibilitat amb sistemes relacionals (sense transaccions, sense restriccions, sense vistes, . . .)
  On està el truc? I ACID és pessimista.
  I BASE es optimista, i resulta no ser-ho en excés: es pot gestionar de manera realista.

Tecnologies i Implementacions NoSQL

- BigTable, Cassandra, HBase

Aquestes són bases de dades de columnes, dissenyades per optimitzar el maneig de grans volums de dades. Són particularment útils en entorns on les lectures i escriptures són massives i on l'eficiència de l'espai i la velocitat d'accés són crítics.

  - BigTable: Desenvolupat per Google, aquest sistema gestiona el emmagatzematge de dades en columnes enlloc de files, permetent una eficient recuperació i anàlisi de grans conjunts de dades distribuïdes.
  - Cassandra: Originada a Facebook i ara un projecte de top de l'Apache Software Foundation, combina el maneig de dades en columnes amb una alta escalabilitat i disponibilitat sense un únic punt de fallida.
  - HBase: També un projecte d'Apache, és una base de dades distribuïda que suporta dades estructurades i que també es basa en el model de BigTable de Google.

- MongoDB, CouchDB

Aquestes són bases de dades orientades a documents que faciliten el treball amb formats de documents com JSON, permetent un esquema dinàmic que fa que l'agregació i la recuperació de dades siguin flexibles i eficients.

  - MongoDB: És una de les bases de dades NoSQL més populars que ofereix alta performance, alta disponibilitat, i escalabilitat fàcil. Suporta un model de dades ric que permet documents incrustats i arrays.
  - CouchDB: Utilitza JSON per a l'emmagatzematge de dades, JavaScript com a llenguatge de consulta, i HTTP com a API, facilitant així la integració amb aplicacions web.

- Redis, DynamoDB

Aquestes bases de dades estan optimitzades per operacions clau-valor, les quals són ideals per a sessions, caches, i aplicacions que requereixen un accés extremadament ràpid a dades.

  - Redis: És una base de dades en memòria que ofereix estructures de dades en clau-valor. És coneguda per la seva velocitat i és àmpliament utilitzada per caching.

    - DynamoDB: Servei de base de dades NoSQL ofert per Amazon Web Services, que ofereix un rendiment de milisegons a qualsevol escala i és completament gestionat, eliminant la necessitat de manejar el hardware o les instàncies de servidor.
- Neo4j, FlockDB

Aquestes són bases de dades de grafs dissenyades específicament per manejar dades i consultes que involucren relacions complexament interconnectades, essent ideals per a anàlisi de xarxes socials, recomanacions, i altres aplicacions on les relacions són tan importants com les dades mateixes.

    - Neo4j: És una base de dades de grafs que proporciona un emmagatzematge eficient de dades de grafs i facilita consultes que impliquen navegació profunda de la relació.
    - FlockDB: Desenvolupada per Twitter, està optimitzada per a emmagatzemar relacions socials a gran escala, com qui segueix a qui.