

En el tema d'hem fet el primer contacte amb les Bases de Dades NoSQL. En aquella ocasió es tractava de Bases de Dades Orientades a objectes, que ens permetien guardar els objectes de forma cómoda, i sense desfaseament objecte-relacional, ja que el que guardavem era directament objectes. Per contra, se'n complicava la manera de fer consultes, ja que no disposavem d'un llenguatge tan potent com el SQL.

En el tema d'hem fet un altre contacte amb les Bases de Dades per a guardar documents XML, de forma eficient i fàcil de recuperar posteriorment la informació, amb consultes potents. Concretament veurem xslt.

Pots en definir els dos models no sols més que dos després concrets. El terme **NoSQL**, **Not Only SQL** té una gran extensió, tot i que en definitiva Bases de Dades que no estan basades en el Model Relacional, i que en determinades ocasions poden ser més eficients que les Bases de Dades Relacionals precisament per fugir de la rigidesa que proporciona aquest model, amb dades tan ben estructurades. El terme clau és "en determinades ocasions", és a dir, per a evitarr un determinat tipus d'informació. Així ens trobarem daltres formes de Bases de Dades NoSQL, dependent del tipus d'informació que volem guardar.

- **Bases de Dades Orientades a Objectes**, per a poder guardar objectes, com per exemple **DBObject**.
- **Bases de Dades Orientades a Documents** (o simplement Bases de Dades Documentals), per a guardar documents de determinat tipus: XML, JSON, Per exemple **xslt** que guarda documents XML, o **MongoDB** que guarda la informació en un format similar a JSON. Menció especial farem de **Firebase**, una Base de Dades que en temps real permet sincronitzar amb el servidor les dades locals. Utilitza també el format JSON.
- **Bases de Dades Clau-Valor**, per a guardar informació de forma molt serialitzada, guardant únicament la clau (el nom de la propietat) i el seu valor.
- **Bases de Dades Multivalor**, que permeten guardar moltes dades de nodes i arrelades que comuniquen entre si (compartiran) els nodes.
- **Bases de Dades Orientades a Columnes**, que té una estructura pareguda a les taules del Model Relacional, però orientada a les columnes (o famílies de columnes), de manera que un grup de columnes es guarda en el mateix bloc). Un exemple és **Cassandra**.

Això tpus com les **Bases de Dades Multivalor**, les **Bases de Dades Tabulars**, ...

En aquest tema només veurem, a part de l'esmentada **Firebase**, les Bases de Dades Clau-Valor, per la seua senzillesa, i una altra Base de Dades, **MongoDB**, per la seua utilització actual. Com que ja hem vist una Orientada a Objectes i en el tema següent veurem una Orientada a Documents XML, la visió global és suficientment extensa.

Firebase

Firebase és un servei de backend, és a dir del costat del servidor, que ofereix la possibilitat de guardar les dades d'aplicacions web i de dispositius mòbils, amb la particularitat que en temps real es poden actualitzar les dades modificades des d'un dels clients connectats a tots els altres que estiguin connectats.

Ofereix una sèrie de serveis, entre els quals podem destacar:

- **Firebase Auth - Autenticació d'usuari:** permet gestionar la validació d'usuari comodament. Inclou l'autenticació per mig de Facebook, GitHub, Twitter i Google, i també la validació utilitzant compte de correu, guardant-se la contrasenya en Firebase, amb la qual cosa ens allibera d'haver de guardar-les al nostre dispositiu.
- **Realtime Database - Base de Dades en Temps Real:** permet guardar unes dades en el nostre, que estaran sincronitzats a totes les aplicacions (clients) connectats. No és realment una Base de Dades, tan sols un document JSON.
- **Cloud Firestore - Base de Dades en documents:** permet guardar documents de tipus JSON, i per tant ja es pot considerar una Base de Dades.
- **Firebase Storage:** permet guardar arxius per a les nostres aplicacions, com poden ser audios, vídeos, imatges, etc.

En aquest tema ens fixarem sobretot en el segon i tercer servei, els de Base de Dades. Hem de recordar, que la **Realtime Database** no és una Base de Dades completa. Tan sols és un document JSON el que podem guardar, així si, tan sengles com volguem. Es tracta per tant d'una Base de Dades NoSQL, o millor dit una mini Base de Dades NoSQL. Això si, que permet sincronitzar les dades en temps real en tots els dispositius connectats.

En canvi el **Cloud Firestore** si que és una Base de Dades completa.

2.1 Creació d'una aplicació

La Base de Dades, que hem quedat que poden ser un document únic JSON (en el cas de Realtime Database, i tot un conjunt en el cas de Cloud Firestore), està associada a una aplicació. Crearem des de l'entorn de Firebase una nova aplicació, la referència de la qual és la que utilitzarem en l'aplicació web o aplicació mòbil. Així doncs, podem crear-nos unes quantes aplicacions Firebase, i en cadauna guardarem una Base de Dades. Ens farà fàcil autenticar-nos amb un compte de Google. Utilitzarem el compte corporatiu de IES El Camí de, de a dir xxx@ieselcamino.es.

El procés de creació el farem des de l'entorn de Firebase: <https://firebase.google.com>

El següent vídeo mostra el procés de creació d'una aplicació.

En les últimes versions de Firebase la primera vegada que s'accedeix a la Base de Dades Firebase acabada de crear, pregunta per les regles (rules) d'accés a aquesta. Inicialment tenrem l'opció modo de prova, en el qual tot el món pot accedir a les dades. Evidentment no es poden deixar aquestes regles de forma definitiva, però per a començar a provar, està bé. De fet, en l'última versió ho deixarà en mode de prova durant un mes. De tota manera, si ho volguem allargar, afirem a la configuració dels rules.



2.2 Realtime Database (RD)

Encara que la finalitat és utilitzar-los del nostre entorn de programació, més concretament Android, sempre ens arribà bé "tocar" les dades directament des d'un entorn propi.
L'entorn que ens ofereix **Firebase** serà suficient. Podrem visualitzar les dades que tenim introduïdes en totes les nostres aplicacions, i també editar-les, introduir, modificar i elaborar.

Recordem que podem utilitzar 2 versions:

- Realtime Database
- Cloud Firestore

Farem especial menció al fet que el que guardem és un document JSON. Fins i tot ens el podem baixar (exportar) o pujar un document nou (importar). Això si, l'operació d'importació destrueix les dades anteriors, mostra de que només podem guardar un document JSON.

Aquí es el contingut del fitxer `Empleats.json` que s'introduix en el vídeo. Està format per una línia de lectura, però en realitat no importaria que estiguera tot seguit.

```
"employees": [
    {
        "name": "Andrea",
        "surname": "Garcia",
        "edad": 10,
        "seu": 32,
        "acum": 1000
    },
    {
        "name": "Sergant",
        "surname": "Garcia",
        "edad": 20,
        "seu": 28,
        "acum": 1200
    },
    {
        "name": "Lidia",
        "surname": "Garcia",
        "edad": 10,
        "seu": 12,
        "acum": 1100
    },
    {
        "name": "A",
        "surname": "Garcia",
        "edad": 10,
        "seu": 12,
        "acum": 1500
    }
]
```

Ja heu vist que en importar `Empleats.json` s'han perdut les altres dades. Poseu una altra vegada la parola clau-vàlor `a1` (per exemple amb el valor `Hola`) ja que per ser molt senzilla l'estructura l'utilitzarem en alguns exemples. Així ens queda l'estructura per als posteriors exemples:



2.2.2 RD: Utilització des de Java

El nostre objectiu final d'accedir a Firebase serà des d'Android. Mirarem primer l'accés des de Java, que és l'enfront que utilitzem en el present mòbil. Passar després a Android serà una cosa senzilla.

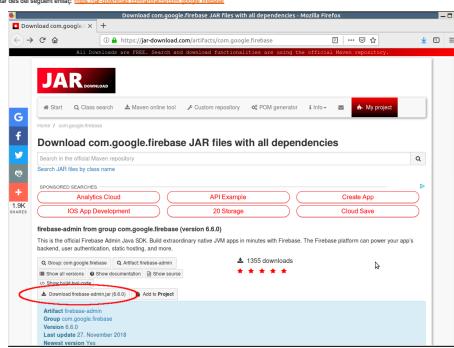
Per als exemples i exercicis d'aquesta part, ens crearem un projecte anomenat Tema7_1_Firebase, amb els paquets Exemples_RealtimeDatabase, Exemples_CloudFirestore i Exercicis.

Drivers necessaris

L'acés des de Java a Firebase no és senzill. És més complicat que accedir des d'Android, en contra del que cabria esperar.

Concretament ens haurem de baixar molts .jar. En canvi en Android ho vindrem tot disponible, ja que els drivers estan incorporats.

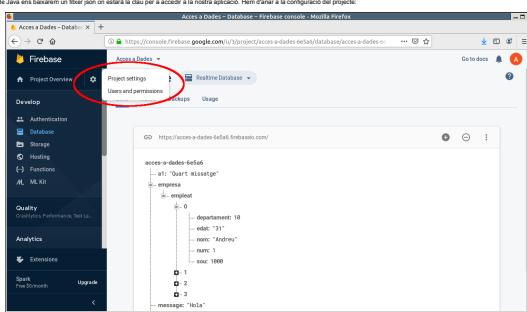
Són tants els drivers que ens construïm una llibreria per a poder incorporar-los després comodament als nostres projectes. Ens els podem baixar des del següent enllaç: [http://jar-download.com/jar/com.google.firebaseio.firebaseio-admin/8.8.2/jar](http://jar-download.com/jar/com.google.firebaseio/firebase-admin/8.8.2/jar)



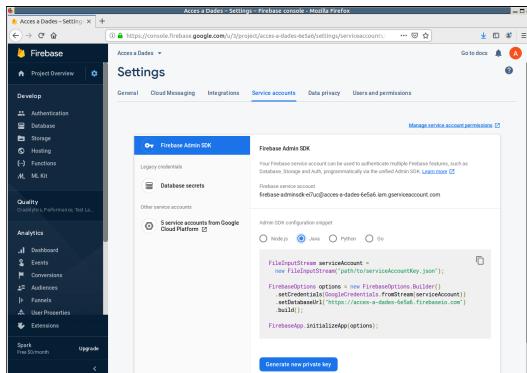
Una vegada baixat i descomprimint el contingut en una carpeta, construïm una llibreria anomenada **Firebase** que incorporate tots els jar.

Configuració

El primer que hem de fer és preparar el nostre projecte per a que puga accedir a l'aplicació que hem creat en Firebase. En l'entorn de Java baixarem un **firebase.json** on establirà la clau per a accedir a la nostra aplicació. Hem d'anar a la configuració del projecte:



I dins de la configuració arribar a la pestanya **Cuentas de servicio (Service Accounts)**. Abi'reuem uns exemples d'utilització (a nosaltres ens interessa Java), i bain de tot un botó per a **Generar una nova clau privada**.



Era baixar un **firebase.json** que haurem de guardar a l'àrea del projecte. Després, com data l'exemple, col·loquem el següent per a un accés correcte:

```
FileInputStream serviceAccount = new FileInputStream(new File("keyfile.json"));
FirebaseOptions options = new FirebaseOptions.Builder()
.setCredentials(FirebaseCredentials.fromStream(serviceAccount))
.setDatabaseUrl("https://acces-a-dades.firebaseio.com").build();
FirebaseApp.initializeApp(options);
```

No us oblideu de substituir el nom del **keyfile.json**. Tamporà heu de tenir en compte que la URL de la base de dades serà diferent per a cada cas de nosaltres.

Referència a la Base de Dades i a les dades concretes a les quals volem accedir

Era haurem de crear un objecte **Firebasedatabase**, que serà una referència a tota la Base de Dades:

```
Firebasedatabase database = Firebasedatabase.getInstance();
```

A part d'ella podrem fer referència a una paràula clau-valor que estiga a l'arrel, com quan havíem creat a **al** (però recordeu que ara no existeix):

```
final DatabaseReference refAl = database.getReference("al");
```

observeu com no es cap impediment que ens troquem a una branca de l'arrel que no ha de canviar en cap moment després d'inicialitzar-la. Si que podem fer canviar el seu valor, però la referència sempre ha d'apuntar al mateix lloc.

També podem fer referència a una parèlla clau-valor que no estige en l'arrel de la Base de Dades. Senceraament posarem la ruta des de l'arrel. Per exemple, per a accedir al nom del primer empleat de l'empresa que fermem guardat, ho farem així:

```
final DatabaseReference refEmployee = database.getReference("empresa/empleat/0/nom");
```

En els casos anteriors hem optat per agafar parèlles clau-valor, bé a l'arrel o més cap a dins de l'estructura JSON. Però en definitiva una parèlla clau-valor.

Tamporà podem optar per agafar l'estructura JSON i treballar amb ella, com van fer en el Tema 3, quan vam treballar amb l'estructura JSON.

```
final DatabaseReference empresa = database.getReference("empresa");
```

D'aquesta manera, en el qual agafem tota l'estructura, tindrem dues maneres de treballar posteriorment per a accedir més avall en l'estructura:

- Passar-la a objectes i arrays de JSON, i treballar com van fer en el Tema 3. Molt comòd, sobretot quan es tracta d'operacions de lectura.
- Treballar directament amb mètodes de Firebase que ens permeten accedir bé a tots els fils d'una estructura, bé a un fill en concret

Ho mostrarem en els exemples posteriors.

Guardar dades

Disposen del mètode `setValue()` de la referència a la dada a la que volem accedir. Accepta 2 paràmetres:

- El primer és el valor que volem introduir.
- El segon és un listener per a poder sintonitzar. Era fer falla sincronitzar únicament en els programes mode text serialitzat que ten. Quan fem els gràfics no farà falla, i posarem null

Si per exemple vulguem guardar en la variable `a`, de forma senzilla ho farem així (ens funciona en els gràfics):

```
refA.setValue("valor per a a", null);
```

En l'operació de guardar si la pàgina clau-vàlor on se va a guardar exista, doncs modifiquar el valor. I si no existeix, la crea.

En el cas que vulguem guarir en un filer, són més avançat en estructura JSON, disposen del mètode `child()`, que ens permet anar a un determinat filer, a l'estil de la segona manera descrita en el subpunkt anterior. Per exemple si vulguem canviar l'edat del primer empleat, l'estructura per a ambar serà: `empresa->empleat->0->edad`

```
empresa.child("empleat").child("0").child("edad").setValue("33", null);
```

Si no existeix abans qualsevol dels nodes de l'estructura, o creua. Fins i tot, se'n crearan abans de la sentència anterior `empresa`, doncs creara `empresa`, i `ids del empleat`, i `ids del edat`, amb el valor 33.

També podem parar tot un objecte, i es convertir en estructura JSON, però ens ho deixem per a més avant.

Potser com havíem dit abans, s'ha de sincronitzar amb Firebase, des del programa Java ens hem esperat a aquella sincronització, si no no ens funcionarà. En el segon parametre ens posen un listener. Aquest exemple ja està complet:

```
import java.io.FileInputStream;
import java.io.IOException;
import java.util.concurrent.CountDownLatch;

import com.google.auth.oauth2.GoogleCredentials;
import com.google.firebaseio.FirebaseApp;
import com.google.firebaseio.FirebaseOptions;
import com.google.firebase.database.DatabaseReference;
import com.google.firebaseio.firebaseio.database.DatabaseReference;
import com.google.firebaseio.firebaseio.database.FirebaseDatabase;

public class ProvaFirebaseGuardar {
    public static void main(String[] args) throws InterruptedException, IOException {
        FileInputStream serviceAccount = new FileInputStream("acces-a-dades-45ad4.firebaseio-adminsdk-e17uc-8f5d9262021.json");
        FirebaseOptions options = new FirebaseOptions.Builder()
                .setCredentials(GoogleCredentials.fromStream(serviceAccount))
                .setDatabaseUrl("https://acces-a-dades-45ad4.firebaseio.com").build();

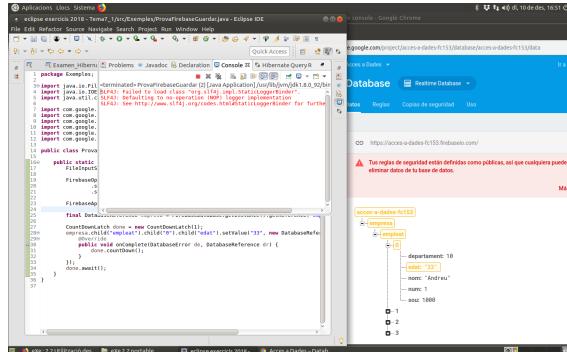
        FirebaseDatabaseApp.initializeApp(options);

        final DatabaseReference empresa = FirebaseDatabase.getInstance().getReference("empresa");

        CountDownLatch done = new CountDownLatch(1);
        empresa.addListenerForSingleValueEvent(new ValueEventListener() {
            @Override
            public void onDataChange(DataSnapshot dataSnapshot) {
                done.countDown();
            }
        });
        done.await();
    }
}
```

Recordiu que heu de canviar el nom del fitxer json a la URL per les vostres.

I aquest és el resultat d'executar-lo. La dreta podem veure com s'està modificant la dada que pretenim:



Pot com comentarem, en el cas de les aplicacions gràfiques resulta més senzill, ja que no haurem d'esperar expressament a la sincronització, sinó que l'aplicació es queda en marxa, i per tant no hi haurà problema.

Ho practicarem en un exemple nou, on arribarem a construir un Xet, i ens servira per a practicar totes les coses que us volem mostrar en Realtime Database de Firebase.

Aquest és l'aspecte del programa:

```
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Container;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.FileInputStream;
import java.io.IOException;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.JTextField;
import javax.swing.SwingConstants;
import javax.swing.WindowConstants;

import com.google.auth.oauth2.GoogleCredentials;
import com.google.firebaseio.FirebaseApp;
import com.google.firebaseio.FirebaseOptions;
import com.google.firebase.database.DataSnapshot;
import com.google.firebase.database.DatabaseError;
import com.google.firebase.database.DatabaseReference;
import com.google.firebaseio.firebaseio.ValueEventListener;
import com.google.firebaseio.firebaseio.ValueListener;

public class Pantalla_CrearAt extends JFrame implements ActionListener {
    private static final long serialVersionUID = 1L;
    JLabel et1ultimoMissatge= new JLabel("Últim missatge: ");
    JLabel ultimoMissatge= new JLabel();
    JTextField area = new JTextField();
    JLabel etIntroducioMissatge = new JLabel("Introduix missatge:");
    JButton enviar = new JButton("Enviar");
    JTextField missatge = new JTextField("Missatge");

    //En iniciar posa en el contenidor per als elements anteriors
    public JPanel panel1= new JPanel(new BorderLayout());
    Jframe pant1Incipit = this;
    this.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
    this.setSize(300, 300);
    this.setLayout(null);
    this.setVisible(true);
    // contenidor per als elements
    //Hi ha un titol. Panel de dalt ultim missatge. Panel central: tot el xat
    JPanel panel1_1 = new JPanel(new GridLayout());
    panel1_1.add(et1ultimoMissatge);
    panel1_1.add(ultimoMissatge);
    panel1_1.add(area);
    getContentPane().add(panel1_1, BorderLayout.NORTH);

    JPanel panel1_2 = new JPanel(new BorderLayout());
    panel1_2.add(etIntroducioMissatge, BorderLayout.NORTH);
    area.setForeground(Color.blue);
    area.setEditable(true);
    area.setLineWrap(true);
    panel1_2.add(missatge, BorderLayout.CENTER);
    getContentPane().add(panel1_2, BorderLayout.CENTER);

    JPanel panel1_3 = new JPanel(new BorderLayout());
    panel1_3.add(et1ultimoMissatge);
    panel1_3.add(enviar, BorderLayout.EAST);
    panel1_3.add(missatge, BorderLayout.SOUTH);
    getVisible(true);
    enviar.addActionListener(this);

    FileInputStream serviceAccount = new FileInputStream("acces-a-dades-45ad4.firebaseio-adminsdk-e17uc-8f5d9262021.json");
    FirebaseOptions options = new FirebaseOptions.Builder()
            .setCredentials(GoogleCredentials.fromStream(serviceAccount))
            .setDatabaseUrl("https://acces-a-dades-45ad4.firebaseio.com").build();

    FirebaseDatabaseApp.initializeApp(options);

    // Exemple de llistar de lectura deixa addListenerForSingleValue()
    // Per a posar el títol. Sobre només
    // Exemple de llistar de lectura continua addValueEventListener()
    // Per a posar l'últim missatge registrat. Sobre al
    // Exemple de llistar d'una llista addChildEventListener()
    // Per a posar tota la llista de missatges. Sobre tot

}

@Override

public void actionPerformed(ActionEvent e) {
    if (e.getSource() == enviar) {
        // exemple de guardar dades sense haver d'esperar per ser una aplicació gràfica
    }
}
```

```
// Per a guardar dades. Sobre al, i després sobre la llista val
}

}

Recordeu que heu de canviar el nom del fitxer json i la URL per les vostres. Ho podeu copiar de l'exemple anterior
I aquest és el programa principal:
```

```
import java.io.IOException;
public class CreateXat {
    public static void main(String[] args) throws IOException {
        final Pantalla_CrearXat finestra = new Pantalla_CrearXat();
        finestra.iniciar();
    }
}
```

Observeu que ja tenim collocades en l'anterior programa les dades de connexió:

```
FileInputStream serviceAccount = new FileInputStream("acces-a-dades-6e1a6.firebaseio-adminsdk-e7uc-8f5d926921.json");

FirebaseOptions options = new FirebaseOptions.Builder()
    .setCredentials(GoogleCredentials.fromStream(serviceAccount))
    .setDatabaseUrl("https://acces-a-dades-6e1a6.firebaseio.com").build();

FirebaseApp.initializeApp(options);
```

I tornem a insistir en què heu de canviar la referència al fitxer JSON i la URL.

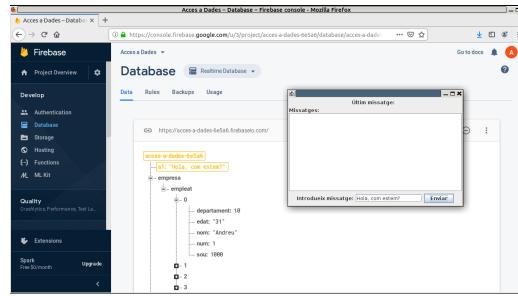
Per a guardar les dades, en aquest exemple de moment guardarem en la clau de Firebase at i en el moment de apretar el botó de baix d'Enviar. No farà falta mutar cap listener per veure si ja hem acabat; ja que el programa continua en marxa.

```
public void actionPerformed(ActionEvent e) {
    if (e.getSource() == enviar) {
        // Exemple de guardar dades sense haver d'esperar per ser una aplicació gràfica
        // Per a guardar dades. Sobre al, i després sobre la llista val
        final DatabaseReference refAl = FirebaseDatabase.getInstance().getReference("al");
        refAl.setValue(message.getText(), null);
    }
}
```

Observeu que ara queda molt senzilla la semàntica de guardar; i no ha fet falta ficar cap listener en el segon paràmetre, sinó null, per estar en una aplicació gràfica:

```
refAl.setValue(message.getText(), null);
```

El resultat seria aquest, on es veu la modificació de la dada que volem:



Recuperar dades

La lectura de dades és més complicada que l'escriptura. Es té bona part per culpa de la "inconsistència" de les dades que obtenim. Per això no existeix un mètode tan senzill com el `getValue()`. La lectura s'ha de montar sempre amb `Listeners`, que es queden escoltant si hi ha alguna actualització de la dada registrada. Recordeu que és de la dada registrada, no de tota la Base de Dades.

Potom mutarem els tipus de `Listeners`, però el seu funcionament serà similar:

- Els que només escolten per a llegir les dades al principi, i no esperaran per a posterior canvi en les dades (i per tant no consumiran tants recursos): `addListenerForSingleValueEvent()`
- Els que es queden escoltant tota l'història: `addValueEventListener()`

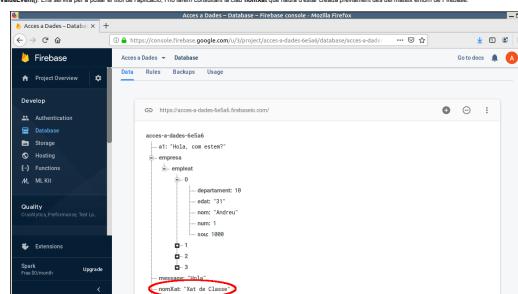
En ambdós casos obtindrem com a paraula un `DataSnapshot` (copia) de la dada registrada. I d'aquest tipus, `DataSnapshot`, que té el mètode `getValue()` per a accedir a la dada. Ambdós tipus de `Listeners` tenen un tractament absolutament similar, únicament amb la diferència abans esmentada que el segon està sempre escoltant, i el primer només escolta una vegada al principi.

El mètode `getVal()` admet un paràmetre que serà la classe del tipus que volen obtenir. Podem posar les següents:

- `String.class`: i aleshores el que obtindrem l'interpretarà com un String
- `Double.class`: i aleshores obtindrem un valor real de doble precisió
- `Boolean.class`: i aleshores obtindrem un valor boolean
- També es poden posar classes per a obtenir tot un objecte (Map) i per a una llista (List). Fins i tot es podria intentar a posar una classe definida per nosaltres. Però amb els anteriors nosaltres en trobem prou

addListenerForSingleValue()

En aquest primer exemple anem a agafar una única vegada la dada que ens interessa. I per tant utilitzarem `addListenerForSingleValueEvent()`. Ens servirà per a posar el títol de l'aplicació. I ho farem consultant la clau `nomXat` que havíem d'estar creada prèviament del mateix entorn de Firebase.



Modifiquem el fragment de programa marcat pel comentari, i el que fem després per a llegir només una vegada.

```
// Exemple d'escuchar la dada anota addListenerForSingleValue()
// Per a posar el títol. Sobre nomXat
final DatabaseReference nomXat = FirebaseDatabase.getInstance().getReference("nomXat");

nomXat.addListenerForSingleValueEvent(new ValueEventListener() {
    @Override
    public void onDataChange(DataSnapshot dataSnapshot) {
        String titol = dataSnapshot.getValue(String.class);
        JOptionPane.showMessageDialog(null, titol);
    }

    @Override
    public void onCancelled(DatabaseError error) {
    }
});
```

Aquest és el resultat, i quan l'executeu observareu que tarda un poc en mostrar el titol. Es perquè ho està llegint de Firebase.



addValueEventListener()

Ara veureu un exemple per a l'altra mètode, el `addValueEventListener()`, que és el que es queda escoltant tota la dada.

Concretament el que farem es escuchar tota la extensió per si es produeix algun canvi en la clau at. Si es produeix aquest canvi, modificarà el valor del JLabel `ultimoMensaje` que està dat.

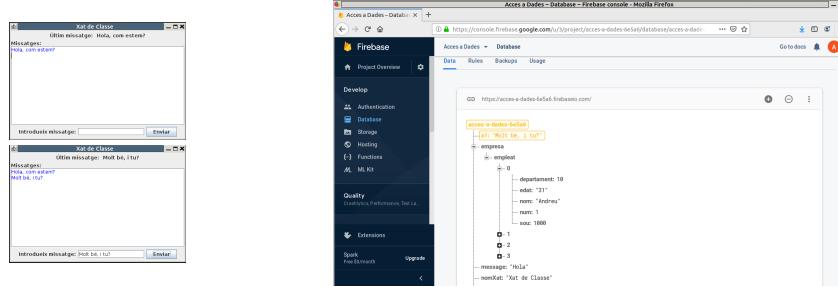
També alegrem el missatge al JTextField, per a que tinga aparença de xat, encara que després ho modifiquem per a millorar-ho. Ho hem de collocar on està marcat pel comentari

```
// Exemple de escuchar continua addValueEventListener()
// Per a posar l'últim missatge registrat. Sobre al
final DatabaseReference ultim = FirebaseDatabase.getInstance().getReference("al");

ultimo.addValueListener(new ValueEventListener() {
    @Override
    public void onDataChange(DataSnapshot dataSnapshot) {
        ultimoMensaje.setText(dataSnapshot.getValue(String.class));
        area.append(dataSnapshot.getValue(String.class) + "\n");
    }

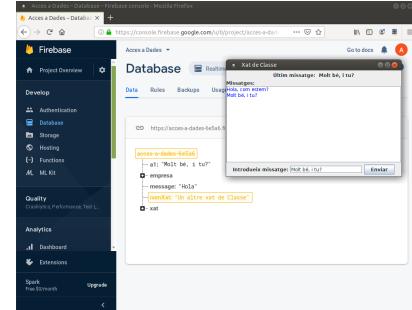
    @Override
    public void onCancelled(DatabaseError error) {
    }
});
```

L'execució serà com la de la pantalla de dalt a l'esquerra. Però si es produeix algun canvi (com es mostra en la pantalla de la dreta), aquest canvi es reflectirà automàticament tant en el JLabel de dalt com en el TextView, tal com es mostra en la imatge de baix a l'esquerra:



Com veieu ha estat molt fàcil construir una espècie de xat. Ara millorarem aquest xat.

Això sí, una vegada està en marxa el programa del xat, per més que canviem nomXat, que es traslladaria al títol de la finestra, aquest títol ja no canviaria perquè recordem que es fa una única lectura.



Tractament de llistes

Per a explicar millor el tractament de llistes, cream un altre referència a una clau que representarà una llista de missatges. Cada missatge constarà d'un nom i un contingut, i així usarem també el tractament d'objectes.

El primer que ens haurà de fer és la referència a aquesta nova clau, que l'anomenarem xat (no està creada encara en la Base de Dades):

`final DatabaseReference xat = FirebaseDatabase.getInstance().getReference("xat");`

Anar alegant elements a la llista, ho podrem fer a mà, posant nodes individuals, ja que hem vist que la manera de representar en Firebase una llista són fils únicament amb clau; que seria el subíndex.

Per tant una manera d'afegir el primer missatge del xat, seria amb l'index 0. Possem aquest codi quan aprofitem el clic (no hem fet de moment el fet de guardar en xat, per a que mentre fem les proves es veja tot el xat):

```
public void actionPerformed(ActionEvent e) {
    if (e.getSource() == enviar) {
        // Exemple de guardar dades sense tenir de esperar per fer una aplicació gràfica
        // Per a fer això caldrà que el dispositiu on es executa l'aplicació tinga accés a Internet
        final DatabaseReference refXat = FirebaseDatabase.getInstance().getReference("xat");
        refXat.setValue(message.getText(), null);
        DatabaseReference refContingut = FirebaseDatabase.getInstance().getReference("xat");
        refContingut.child("0").child("nom").setValue("User1");
        refContingut.child("0").child("contingut").setValue(message.getText());
    }
}
```

Quan aquesta s'actualitzarà la Base de Dades d'aquesta manera:



Però aquella manera d'introduir en la llista acaba per ser molt poc pràctica. Si s'afegeix un segon missatge (nom i contingut) li hauríem de posar com a index 1. No és viable.

Podriem dues opcióies per a positionar els ïndexs:

• Podriem posar un comptador per a saber quin índex toca inserir en cada moment, cosa també molt poc pràctica perquè si l'aplicació està instal·lada en més d'un dispositiu, podrà haver col·lidit en el número d'índex.

• Podriem mirar quin és l'últim índex immediat, per a incrementar-lo en una unitat. Però això suposa la llegi de la Base de Dades, i com hem vist abans exposa muntar un Listener, segurament d'est d'un únic. Es pot fer, però no es comode.

Per a establir-nos aquella feina, Firebase ens proporciona un mètode per a afegir un nou element a una llista, el mètode `push()`. Introduïm un nou element a la llista, i l'posa com a index un número general de manera que no es repetirà mai. L'única preocupació que hem de tenir és guardar aquest index (`getKey()`), per a poder posar-lo com a clau:

```
public void actionPerformed(ActionEvent e) {
    if (e.getSource() == enviar) {
        // Exemple de guardar dades sense haver d'esperar per fer una aplicació gràfica
        // Per a fer això caldrà que el dispositiu on es executa l'aplicació tinga accés a Internet
        final DatabaseReference refXat = FirebaseDatabase.getInstance().getReference("xat");
        refXat.push().setValue(message.getText(), null);
        DatabaseReference refContingut = FirebaseDatabase.getInstance().getReference("xat");
        String clau = refContingut.push().getKey();
        refContingut.child(clau).child("nom").setValue("User1");
        refContingut.child(clau).child("contingut").setValue(message.getText());
    }
}
```

I el resultat de aquest:



Com veieu és una cadena molt llarga que no es repetirà mai.

Ara hem afegeix un missatge, però ara ho completem més, i solucionem de pas algun problema que podem haver tingut.

Crearem una classe anomenada `Message`, que inclourà els propietats nom i contingut. Crearem un objecte `Message` amb uns nous valors, i veurem que el podem guardar perfectament. Per a aquest exemple segurament no valdrà la pena l'esforç, però es pot veure la utilitat per a objectes més complexos.

Primer definim la classe. El mètode és que el guardem com una classe nova, no a dir com a `Message.java`:

```
public class Message{
    public String nom;
    public String contingut;
    public Message(){
    }
    public Message(String nom, String contingut){
        this.nom=nom;
        this.contingut=contingut;
    }
}
```

Millor tenir tots els getters i setters.

Per a guardar, cor hoarem aquestes sentències entre les actions del clic del botó, com en les altres ocasions. Observeu com ara ni tan sols es fa guardar en una variable el `getKey()`, ja que es fa tot només en una operació:

```
public void actionPerformed(ActionEvent e) {
    if (e.getSource() == enviar) {
        final DatabaseReference refXat = FirebaseDatabase.getInstance().getReference("xat");
        refXat.push().setValue(message.getText(), null);
        final DatabaseReference xat = FirebaseDatabase.getInstance().getReference("xat");
        Message m = new Message("User1", message.getText());
        xat.push().setValue(m,null);
    }
}
```

El resultat seria el mateix que en l'ocasió anterior:



Ara només ens falta el tractament de lectura de les llistes.

addChildEventListener()

Podrem muntar un Listener com en les altres ocasions, però Ara disposarem d'una altra listeners que seràn acoplats més junt que s'activen quan hi ha modificacions en algun element de la llista. En l'exemple només utilitzarem el de creació d'un nou element, però com veurem també podríem utilitzar els moments de supressió o modificació d'elements.

En tracta del Listener `ChildEventListener`, hem d'utilitzar el mètode `addChildEventListener()` sobre la llista. Veire la implementació dels mètodes: `onChildAdded()`, `onChildChanged()`, `onChildRemoved()` i `onChildMoved()`. Però com comentarem ara només utilitzarem la de creació d'un nou element, i senzillament el mostrarem en el TextView.

Al `datasnapshot` arriba únicament l'element introduït, modificat o esborrat, no tota la llista. Per tant és molt comú. També arriba una referència a l'element anterior com a segon paràmetre, per si hem de fer algun tractament. Aquí ens substituirem l'escriptura que feiem abans del `Textview` (a mi m'havia quedat en la línia 101). Per tant ferem aquesta llista:

```
area.append("\nmissatge: " + dataSnapshot.getValue(String.class) + "\n"); // mostra llista després de l'afegeix
```

I posem el següent on quede marcat pel comentari. Així ens quedarà un rat més "professional"

```
// Exemple de listener d'una llista addChildEventListener()
// Per a posar tota la llista de missatges. Sobre xat
final DatabaseReference xat = FirebaseDatabase.getInstance().getReference("xat");

xat.addChildEventListener(new ChildEventListener() {
    @Override
    public void onChildAdded(DataSnapshot dataSnapshot, String s) {
        area.append(dataSnapshot.child("ultimo").getValue(String.class) + "\n" +
                dataSnapshot.child("ultimo").child("contingut").getValue(String.class) + "\n");
    }

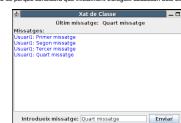
    @Override
    public void onChildChanged(DataSnapshot dataSnapshot, String s) {
    }

    @Override
    public void onChildRemoved(DataSnapshot dataSnapshot) {
    }

    @Override
    public void onChildMoved(DataSnapshot dataSnapshot, String s) {
    }

    @Override
    public void onCancelled(DatabaseError databaseError) {
    }
});
```

Si executem aquest programa, tenint només en el clic del botó l'addició d'un element a la llista i aquest Listener anterior, veurem que inicialment ens apareixeran tots els elements de la llista. Això és perquè considera que inicialment s'afegeix cada un dels elements de la llista, i en el mateix ordre en que estan definits. En aquest instant hem aprofitat per afegeir un quart missatge.



Tot l'exemple

Ara a posar tot l'exemple que aneja tot l'anterior.

Primer la classe `Message.java`:

```
public class Message{
    public String nom;
    public String contingut;

    public Message(){
    }

    public Message(String nom, String contingut) {
        this.nom = nom;
        this.contingut = contingut;
    }

    public String getNom() {
        return nom;
    }

    public void setNom(String nom) {
        this.nom = nom;
    }

    public String getContingut() {
        return contingut;
    }

    public void setContingut(String contingut) {
        this.contingut = contingut;
    }
}
```

Ara el programa:

```
import java.awt.BorderLayout;
import java.awt.Container;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowEvent;
import java.awt.event.WindowListener;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.JTextField;
import javax.swing.WindowConstants;

import com.google.auth.oauth2.GoogleCredentials;
import com.google.firebase.FirebaseApp;
import com.google.firebase.database.DatabaseError;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.ValueEventListener;

public class Pantalla_Creacio extends JFrame implements ActionListener {

    private static final long serialVersionUID = 1L;
    JLabel etiquetaMissatge = new JLabel("Introdueix missatge: ");
    JLabel ultimMissatge = new JLabel();
    JTextField missatge = new JTextField("Introdueix missatge");
    JTextArea area = new JTextArea();
    JButton enviar = new JButton("Enviar");
    JButton introducirMissatge = new JButton("Introdueix missatge");

    // Per a iniciar posem un container per als elements anteriors
    public void iniciar() throws IOException {
        JPanel panel1 = new JPanel(new GridLayout(1, 1));
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setBounds(100, 100, 450, 300);
        this.setLayout(panel1);
        panel1.add(introducirMissatge);
        panel1.add(missatge);
        panel1.add(area);
        panel1.add(etiquetaMissatge);
        panel1.add(enviar);

        //M per a la part del missatge. Panel de dalt: darrere missatge. Panel central: tot el xat
        JPanel panel2 = new JPanel(new BorderLayout());
        panel2.setLayout(new BorderLayout());
        area.setEditable(false);
        jScrollPane1 = new JScrollPane(area);
        panel2.add(jScrollPane1, "Center");
        panel2.add(etiquetaMissatge, "North");
        getContentPane().add(panel2, BorderLayout.CENTER);

        JPanel panel3 = new JPanel(new BorderLayout());
        panel3.setLayout(new BorderLayout());
        panel3.add(introducirMissatge, "North");
        panel3.add(area, "Center");
        panel3.add(etiquetaMissatge, "South");
        getContentPane().add(panel3, BorderLayout.SOUTH);

        setVisible(true);
        enviar.addActionListener(this);
    }

    FileInputStream serviceAccount = new FileInputStream("acces-a-dades-firebase-adminsdk-e17uc-0f5d26921.json");

    FirebaseOptions options = new FirebaseOptions.Builder()
        .setCredentials(GoogleCredentials.fromStream(serviceAccount))
        .setDatabaseUrl("https://acces-a-dades.firebaseio.com").build();

    FirebaseApp.initializeApp(options);

    // Exemple de l'usuari de Doctor senza addListenerForSingleValue()
    // Per a posar el títol. Sobre tot
    final DatabaseReference nomat = FirebaseDatabase.getInstance().getReference("nomat");

    nomat.addListenerForSingleValueEvent(new ValueEventListener() {
        @Override
        public void onDataChange(DataSnapshot dataSnapshot) {
            String s1 = dataSnapshot.get("nom").getValue(String.class);
            String s2 = dataSnapshot.get("contingut").getValue(String.class);
        }
    });

    // Exemple de l'usuari de Doctor continua addValueEventListener()
    // Per a posar l'últim missatge registrat. Sobre al
    final DatabaseReference ultim = FirebaseDatabase.getInstance().getReference("ultim");

    ultim.addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(DataSnapshot dataSnapshot) {
            ultimMissatge.setText(dataSnapshot.getValue(String.class));
        }
    });

    // Exemple de l'usuari de Doctor ultime addCancelled(DatabaseError error) {
    // Exemple de l'usuari d'una llista addChildEventListener()
    // Per a posar el títol. Sobre tot
    final DatabaseReference xat = FirebaseDatabase.getInstance().getReference("xat");

    xat.addChildEventListener(new ChildEventListener() {
        @Override
        public void onChildAdded(DataSnapshot dataSnapshot, String s) {
            area.append(dataSnapshot.child("nom").getValue(String.class)) + "\n";
            dataSnapshot.child("contingut").getValue(String.class)) + "\n";
        }
    });

    // Exemple de l'usuari d'una llista addChildChanged(DataSnapshot dataSnapshot, String s) {
    // Exemple de l'usuari d'una llista addChildRemoved(DataSnapshot dataSnapshot) {
    // Exemple de l'usuari d'una llista addChildMoved(DataSnapshot dataSnapshot, String s) {
    // Exemple de l'usuari d'una llista onCancelled(DatabaseError databaseError) {
    });

    @Override
    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == enviar) {
            // Per a no haver de tancar la finestra sense haver d'esperar per ser una aplicació gràfica
            // Per a guardar dades. Sobre al, i després sobre la llista xat
            final DatabaseReference refat = FirebaseDatabase.getInstance().getReference("ultim");
            refat.setValue(area.getText());
            final DatabaseReference xat = FirebaseDatabase.getInstance().getReference("xat");
            Message m = new Message("usuari1", missatge.getText());
            xat.push().setValue(m);
        }
    }
}
```

```
I per últim el programa principal

import java.io.IOException;
public class CrearXat {
    public static void main(String[] args) throws IOException {
        PantallaCreateXat finestra = new PantallaCreateXat();
        finestra.setVisible(true);
    }
}
```

En el mòdul d'Accés a Dades ens estem plantejant sempre accedir a les dades des de Java. Tanmateix, és en Android on moltes vegades tindrà més aplicació, com és el cas d'accés a Firebase. Per això ho inclourem en aquest tema, en compte de deixar-lo per a un annex.

2.2.3.1 RD-Android: Connexió des d'Android

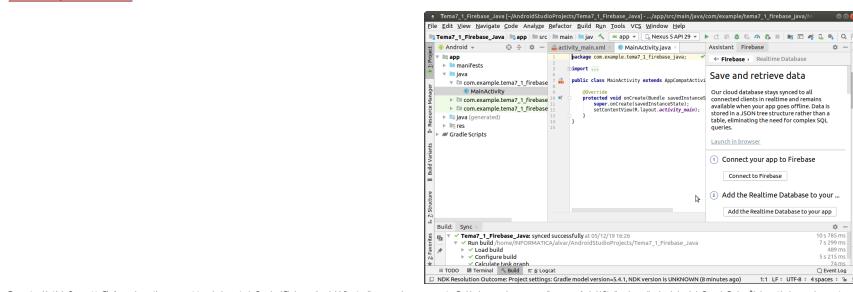
L'accés és extraordinàriament fàcіt gràcies als assistents que ens proporciona el propi Android Studio.

Per a poder provar-lo ens crearem un projecte d'Android anònim per exemple **Tema7_1_FirebaseJava** i **Tema7_1_FirebaseKotlin**.

Al final de tot tenim un vídeo que reposa tots els passos per a poder connectar des de la nostra aplicació d'Android. A continuació els repassarem i expliquem a un a un. Són els que ens marca l'assistent, que haurem d'iniciar sobre el nostre projecte ja creat, i que es crea des de **Tools > Firebase > Realtime Database**.

Observem que quasi tot és idèntic, encara que utilitzem Java o Kotlin. Només farà la diferenciació en l'exemple, on està ja tot el codi.

[Connectar l'aplicació a Firebase](#)



En apretar el botó de **Connect to Firebase**, si no estàvem connectats amb el compte de Google al Firebase se'n obrirà finestra d'un navegador per a conectar. Podrà donar-se el cas que diu que Android Studio va accedir a les dades de la Base de Dades. Obviament ho haurem de permetre.



Una vegada autenticats en Firebase, des de l'entorn d'Android Studio ens ofereix la possibilitat de crear una aplicació nova (una Base de Dades nova) o utilitzar alguna de les ja temes. Utilitzarem la que ens ha servit de prova fins el moment:



Quan heu connectat substituir el botó **Connect to Firebase**, per una etiqueta que dirà **connected**, en verd.

[Afegir la Base de Dades a la nostra aplicació](#)

En aquest segon pas, quan aprem el botó **Add the Realtime Database to your app**, ens dirà els canvis que farà per a incorporar les coses necessàries per a poder connectar.



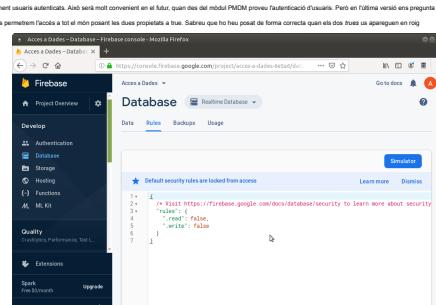
Com veieu es tracta d'inserir les llibreries necessàries de Firebase. En el vídeo del final es mostren imatges amb les dades incorporate.

Igual que abans, substituir el botó **Add the Realtime Database to your app**, per una etiqueta que dirà **Dependencies set up correctly**, en verd. Es una bona guia per saber en quin punt estem.

[Permetre l'accés als usuaris, si es precsa carant les regles d'accés a la Base de Dades](#)

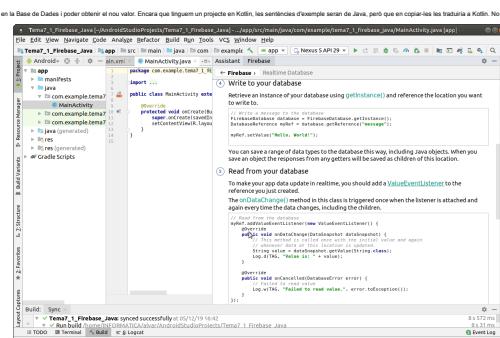
En versions anteriors de Firebase, per defecte les Bases de Dades (aplicacions) de Firebase estaven configurades per a que es connecten únicament usuaris autèntics. Això seria molt convenient en el futur, quan des del mòdul PHCM proveu l'autenticació d'usuaris. Però en l'última versió ens pregunta quan fem el primer accés si volem que es connecten usuaris autèntics, o si permetem que es connecti tot el món.

Sempre ho podem modificar, i així es fa des de la Consola de Firebase de la Base de Dades, en la pestanya **Rules**. En el mòdul d'Accés a Dades permetrem l'accés a tot el mòbil posant les dues projectes a true. Sabreu que ho heu posat de forma correcta quan els dos trues us apareguen en neg



[Copiar les sentències per a escriure i per a llegir \(millor dit per a detectar els canvis en temps real\)](#)

Ens d'u un exemple de les sentències a copiar per a poder guardar una informació a la Base de Dades i també per a detectar un canvi en la Base de Dades per poder obtenir el nou valor. Encara que seguirem un projecte en Kotlin, les sentències d'exemple seran de Java, però que en copiar-les les traduirem a Kotlin. No hem diferenciat per tant Java i Kotlin en aquest moment.



Exemple

A partir del anterior, anem a modificar-lo lleugerament, col·locant un `EditText` on es puga escriure i enviar una informació a la Base de Dades, o quan es modifica en la Base de Dades podem reflectir-ho.

Per a enviar utilitzarem un botó i senzillament quan s'apressa s'envia la informació modificant el valor de `refAt`.

Per a rebre la informació hem d'escollir si hi ha canvis, mitjançant un `ValueEventListener` sobre `refAt`.

Aspecte serà el `ActivityMain.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp"
    tools:context=".MainActivity">
    <EditText
        android:id="@+id/editText"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_margin="16dp"/>
    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/text"
        android:layout_margin="16dp"/>
</androidx.constraintlayout.widget.ConstraintLayout>
```

JAVA

I aquest serà el programa principal en Java:

```
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.EditText;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;
import com.google.firebase.database.ValueEventListener;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        final EditText editText = findViewById(R.id.editText);

        FirebaseDatabase database = FirebaseDatabase.getInstance();
        DatabaseReference refAt = database.getReference("Message");

        final Button btn = findViewById(R.id.button);
        btn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                refAt.setValue(editText.getText().toString());
                editText.setText("");
            }
        });

        refAt.addValueEventListener(new ValueEventListener() {
            @Override
            public void onDataChange(DataSnapshot dataSnapshot) {
                // This method is called once with the initial value and again
                // whenever data at this location is updated.
                String value = dataSnapshot.getValue(String.class);
                Log.d(TAG, "Value is: " + value);
                editText.setText(value);
            }

            @Override
            public void onCancelled(DatabaseError error) {
                // Failed to read value
                Log.w(TAG, "Failed to read value.", error.toException());
            }
        });
    }
}
```

KOTLIN

I aquest és el programa principal per al projecte en Kotlin:

```
import android.support.v7.app.AppCompatActivity
import android.os.Bundle
import kotlin.android.synthetic.main.activity_main.*
import com.google.firebase.database.DatabaseReference
import com.google.firebase.database.FirebaseDatabase
import com.google.firebase.database.DataSnapshot
import com.google.firebase.database.DatabaseError
import com.google.firebase.database.ValueEventListener

class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val refAt = FirebaseDatabase.getInstance().getReference("Message")

        btnEnvia = findViewById(R.id.button)
        refAt.set ValueEventListener {
            refAt.setValue(text.text.toString())
            text.setText("")
        }

        refAt.addValueEventListener(object : ValueEventListener {
            override fun onDataChange(dataSnapshot: DataSnapshot) {
                // This method is called once with the initial value and again
                // whenever data at this location is updated.
                val value = dataSnapshot.getValue(String::class.java)
                text.setText(value)
            }

            override fun onCancelled(error: DatabaseError) {
                // Failed to read value
                Log.w(TAG, "Failed to read value.", error.toException())
            }
        })
    }
}
```

En el següent vídeo es veu tot el procés des del principi per al projecte Kotlin, encara que la clau valor utilitzada es diu `message` en compte de `a1`, i la referència `myRef` en compte de `refAt`:

2.2.3 RD-Android: Accés a les dades

Una vegada vist un exemple, en el qual hem guardat dades i hem vist els canvis en temps real,arem a veure més exemples de com accedir a les dades.

En l'exemple anterior hem accedit a una parella clau-valor situada en l'arrel de la Base de Dades.

Ara veuen algunes exemples de com accedir dins de l'estrucció JSON guardada en Firebase. Com el que voldrem serà mostrar més informació, incorporarem un `TextView` per a mostrar més coses fàcilmént, i deixem el `EditText` i el `Button` per a poder introduir informacions.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/txtAl"
        android:layout_width="wrap_content"
        android:layout_height="36dp"
        android:layout_alignParentTop="true"
        android:layout_alignParentRight="true"
        android:layout_alignParentBottom="true"
        android:layout_constraintHorizontal_bias="0.0"
        android:layout_constraintVertical_bias="0.0"/>
    <EditText
        android:id="@+id/editAl"
        android:layout_width="317dp"
        android:layout_height="45dp"
        android:layout_margin="16dp"
        android:layout_toLeftOf="@+id/btnAl"
        android:layout_toStartOf="@+id/btnAl"
        android:layout_alignBaseline="parent"
        android:layout_alignBottom="parent"
        android:layout_alignTop="parent"
        android:layout_alignEnd="parent"
        android:layout_alignStart="parent"/>
    <Button
        android:id="@+id/btnAl"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:layout_toLeftOf="@+id/editAl"
        android:layout_toStartOf="@+id/editAl"
        android:layout_alignBaseline="parent"
        android:layout_alignBottom="parent"
        android:layout_alignTop="parent"
        android:layout_alignEnd="parent"
        android:layout_alignStart="parent"/>
</androidx.constraintlayout.widget.ConstraintLayout>
```

Així les paraules indicades, que ens serviran per a mostrar cascades de les coses que volem, de forma similar que en Eclipse seran:

- El títol de l'aplicació, que es modificaran amb el `addListenerForSingleValueEvent()`
- Tots els missatges que s'envien, que es podrà modificar amb el `addValueEventListener()`
- El TextView central que coupa quasi tota la pantalla, anomenat `area`, que es modificaran amb la modificaçió de la llista `addChildEventListener()`
- El EditText de baix, anomenat `text`, i el botó que servirà per a afegir un nou element del `set`.

Aquest és l'esquema del programa:

JAVA

```
import android.appCompat.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;
import com.google.firebase.database.ValueEventListener;
import com.google.firebase.database.ChildEventListener;
```

```
public class MainActivity extends AppCompatActivity {
```

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        final TextView txtAl = findViewById(R.id.txtAl);
        final EditText editAl = findViewById(R.id.editAl);
        final Button btnAl = findViewById(R.id.btnAl);
        final EditText text = (EditText) findViewById(R.id.text);
    }
}
```

KOTLIN

```
import android.appCompat.app.AppCompatActivity
import android.os.Bundle
import android.view.View
import android.widget.Button
import android.widget.EditText
import android.widget.TextView
import com.google.firebase.database.FirebaseDatabase
import com.google.firebase.database.DatabaseError
import com.google.firebase.database.ValueEventListener
import com.google.firebase.database.ChildEventListener

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        btnAl.text = "Enviar"
    }
}
```

Referència a la Base de Dades i a les dades concretes a les quals volem accedir

Podem fer referència de forma individual a cascades de les "variables" guardades, és a dir a cascades de les parelles clau-valor. En l'exemple anterior havíem utilitzat aquest:

JAVA

```
FirebaseDatabase database = FirebaseDatabase.getInstance();
// Per a les parelles clau-valor que utilitzarem en l'exemple:
final DatabaseReference refAl = database.getReference("al");
final DatabaseReference nonXat = database.getReference("nonXat");
final DatabaseReference xat = database.getReference("xat");
```

Observem com no és cap impediment que ens toque definir com a final ja que aquesta referència no ha de canviar en cap moment després d'inicialitzar-la. Si que podem fer canviar el seu valor, però la referència sempre ha d'apuntar al mateix lloc.

KOTLIN

L'equivalent en Kotlin serà:

```
val database = FirebaseDatabase.getInstance()
val refAl = database.getReference("al")
val nonXat = database.getReference("nonXat")
val xat = database.getReference("xat")
```

Guardar dades

L'operació de guardar és més senzilla que la de recuperar. Disposam del mètode `setValue()` de la referència a la dada a la que volem accedir. En l'exemple de la pregunta anterior guardarem la dada d'aquesta manera.

Per a JAVA

```
refAl.setValue(text.getText().toString());
```

Per a KOTLIN

```
refAl.setValue(text.text.toString())
```

En l'operació de guardar si la parella clau-valor on es va a guardar existia, doncs modificarà el valor. I si no existia, la creaix.

En el nostre exemple guardarem en el moment d'apretar el botó. Inicialment ho farem sobre `al`, encara que quan vegeu les llistes ho canviarem

JAVA

```
// Exemple de guardar dades. Primera sobre al, i després sobre la llista xat
```

```
}
```

(en el codi original hi ha un error de tipografia)

```
boto.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        refAL.setValue(text.getText().toString());
        text.setText("");
    }
});
```

KOTLIN

```
// Exemple de guardar dades. Primer sobre al, i després sobre la llista xat
boto.setOnClickListener {
    refAL.setValue(text.text.toString())
    text.setText("")
}
```

Recuperar dades

Recordem que podem meter dos tipus de listeners, però el seu funcionament serà similar

- Si es escullen noms de vegada al principi addListenerForSingleValueEvent()
- Si es que es queden escollint tots l'objecte addValueEventListener()

En ambdós casos obtindrem un objecte de la classe **DataSnapshot** (copy de la data registrada). I d'aquest tipus, **DataSnapshot**, si que tenim el mètode **getValue()** per a accedir a la data. Ambdós tipus de listeners tenen un tractament absolutament similar, únicament amb la diferència abans esmentada que el primer està sempre escollint, i el segon només escolla una vegada al principi.

El mètode **get()** sempre ens donarà el contingut de la clau que volem obtenir. Podem posar les següents:

- **String class**: i esdevindrà el que obtindrem d'una pista com un String
- **Double class**: i s'interpretarà com un número real de dades positiu
- **Boolean class**: i s'interpretarà com un valor boolean

- També es poden posar classes per a obtenir un objecte (Map) per a una llista (List). Fins i tot es podrà aniar a posar una classe definida per nosaltres. Però amb els anteriors nosaltres en tindrem prou

Mirem la manera d'utilitzar una altra. Ho apelarem a dues referències diferents, per a que pugueu comprovar que una sempre té efecte, mentre que l'altra només té efecte la primera vegada, i si després es modifiquen les dades, l'aplicació no s'enterà.

addListenerForSingleValueEvent()

En aquest primer exemple anem a afegir una critica regardada la dada que ens interessa, i per tant utilitzarem **addListenerForSingleValueEvent()**. Ens servirà per a posar el títol de l'aplicació. Hi farem consultant la clau **nomXat**, que la tindrem creada des de la pregunta 2.2.2 de la part de Java.

JAVA

```
// Exemple de listener de lectura única addListenerForSingleValueEvent()
// Per a posar el títol sobre nomXat
mentor.addListenerForSingleValueEvent(new ValueEventListener() {
    @Override
    public void onDataChange(DataSnapshot dataSnapshot) {
        String value = dataSnapshot.getValue(String.class);
        setTitle(value);
    }

    @Override
    public void onCancelled(DatabaseError error) {
    }
});
```

KOTLIN

```
// Exemple de listener de lectura única addListenerForSingleValueEvent()
// Per a posar el títol sobre nomXat
mentor.addValueEventListener(object : ValueEventListener {
    override fun onDataChange(dataSnapshot: DataSnapshot) {
        val value = dataSnapshot.getValue(String::class.java)
        ultim.setText(value)
    }

    override fun onCancelled(error: DatabaseError) {
    }
});
```

addValueEventListener()

És la que sempre està escuchant. Per tant qualsevol modificació en la Base de Dades es veurà reflectit, és a dir, canviará el contingut del **TextView** ultim

JAVA

```
// Exemple de listener de lectura continua addValueEventListener()
// Per a posar l'últim missatge registrat. Sobre al
refAL.addValueEventListener(new ValueEventListener() {
    @Override
    public void onDataChange(DataSnapshot dataSnapshot) {
        String value = dataSnapshot.getValue(String.class);
        ultim.setText(value);
    }

    @Override
    public void onCancelled(DatabaseError error) {
    }
});
```

KOTLIN

```
// Exemple de listener de lectura continua addValueEventListener()
// Per a posar l'últim missatge registrat. Sobre al
refAL.addValueEventListener(object : ValueEventListener {
    override fun onDataChange(dataSnapshot: DataSnapshot) {
        val value = dataSnapshot.getValue(String::class.java)
        ultim.setText(value)
    }

    override fun onCancelled(error: DatabaseError) {
    }
});
```

Tractament de llistes

Ja vam veure en la pregunta 2.2.2 de la part de Java que la manera més pràctica d'inserir elements en una llista, per a no haver de dur un manteniment dels index, era amb el mètode **push()**. I que per a no tenir efectes no desejats era millor posar tot l'element d'una vegada. Així, si hem de posar el nom de l'usuari que farà l'entrada del xat i el contingut del missatge, millor crear un objecte que ho contingui tot, i col·locar en la llista aquest objecte.

Crearem per tant una classe anomenada **Message**, que inclourà les propietats **nom** i **contingut**. Crearem un objecte **Message** amb uns nous valors, i veurem que el podem guardar perfectament.

Primer definim la classe. El mètòd és que el guardem com una classe nova, és a dir com a **Message.java** o **Message.kt**

JAVA

```
public class Message{
    public String nom;
    public String contingut;

    public Message(){}
    public Message(String nom, String contingut) {
        this.nom=nom;
        this.contingut=contingut;
    }
}
```

Hauran de tenir també els getters i setters.

KOTLIN

```
class Message(val nom: String, val contingut: String)
```

Per a guardar, col·locarem aquestes sentències entre les accions del OnClickListener del botó, substituint les sentències que modificaven **al**, perquè ara afegeixin a la llista **xat**:

JAVA

```
// Exemple de guardar dades. Primer sobre al, i després sobre la llista xat
boto.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        refAL.setValue(text.getText().toString());

        Message m = new Message("User1",text.getText().toString());
        xat.push(m).setValue(m);
        text.setText("");
    }
});
```

KOTLIN

```
// Exemple de guardar dades. Primer sobre al, i després sobre la llista xat
boto.setOnClickListener {
    refAL.setValue(text.text.toString())
    xat.push().setValue(Message("User1",text.text.toString()))
    text.setText("")
}
```

Ara només ens falta el tractament de lectura de les llistes.

Utilitzarem el Listener **ChildEventListener**, i hem d'utilitzar el mètode **addChildEventListener()** sobre la llista. Voldrà la implementació dels mètodes: **onChildAdded()**, **onChildChanged()**, **onChildRemoved()** i **onChildMoved()**, però nosaltres només utilitzarem **onChildAdded()**.

Al **dataSnapshot** arriba únicament fàlament introduït, modificat o esborrat, no tota la llista. Per tant és molt comòdol. També arriba una referència a l'element anterior com a segon paràmetre, per si hem de fer algun tractament, cosa que en aquest exemple no ens fa falta.

JAVA

```
// Exemple de llistar d'una llista addChildEventListener()
// Per a posar tota la llista de missatges. Sobre xat
xat.addChildEventListener(new ChildEventListener() {
    @Override
    public void onChildAdded(DataSnapshot dataSnapshot, String s) {
        area.append(dataSnapshot.child("nom").getValue(String.class) + ":" + dataSnapshot.child("contingut").getValue(String.class) + "\n");
    }

    @Override
    public void onChildChanged(DataSnapshot dataSnapshot, String s) {
    }

    @Override
    public void onChildRemoved(DataSnapshot dataSnapshot) {
```

```
    }

    @Override
    public void onChildMoved(DataSnapshot dataSnapshot, String s) {
    }

    @Override
    public void onCancelled(DatabaseError databaseError) {
    }
};

KOTLIN

// Exemple de llistar els noms dels clients
// Per a posar tota la llista de més vistes. Sobre tot
val mChildEventListener: ChildEventListener = object : ChildEventListener {
    override fun onChildAdded(dataSnapshot: DataSnapshot, s: String?) {
        area.append(dataSnapshot.child("name").getValue(String::class.java) + "\n")
    }

    override fun onChildChanged(dataSnapshot: DataSnapshot, s: String?) {
    }

    override fun onChildRemoved(dataSnapshot: DataSnapshot) {
    }

    override fun onChildMoved(dataSnapshot: DataSnapshot, s: String?) {
    }

    override fun onCancelled(databaseError: DatabaseError) {
    }
}

Aquí d'ús seria el resultat:
```



Tot l'exemple

Aixem a posar tot l'exemple que anejaig tot l'anterior. Quedarà un poc embolicat, però intentarem explicar després cada eixida per quina causa és.

JAVA

Primer la classe Message.java

```
public class Message{
    public String nom;
    public String contingut;
    public Message(){
    }
    public Message(String nom, String cont){
        this.nom=nom;
        this.contingut=cont;
    }
    public String getNom(){
        return this.nom;
    }
    public String getContingut(){
        return this.contingut;
    }
    public void setNom(String nom){
        this.nom=nom;
    }
    public void setContingut(String contingut){
        this.contingut=contingut;
    }
}
```

I aquest el programa:

```
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;
import com.google.firebase.database.DataSnapshot;
import com.google.firebase.database.DatabaseError;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;
import com.google.firebase.database.ValueEventListener;
import com.google.firebase.database.ChildEventListener;

public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        final TextView ultim = (TextView) findViewById(R.id.ultim);
        final TextView area = (TextView) findViewById(R.id.area);
        final EditText text = (EditText) findViewById(R.id.text);
        final Button bota = (Button) findViewById(R.id.bota);
        bota.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                final String text = text.getText().toString();
                Message m = new Message("User1",text.toString());
                m.setNom("User1");
                m.setContingut(text);
                ultim.setText(m.getContingut());
                text.setText("");
            }
        });
        // Exemple de guardar dades. Primer sobre ai, i després sobre la llista sat
        bota.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                DatabaseReference refai = FirebaseDatabase.getInstance().getReference("ai");
                DatabaseReference refm = FirebaseDatabase.getInstance().getReference("modat");
                refai.setValue(m);
                refm.child("User1").setValue(m);
            }
        });
        // Exemple de guardar dades. Primer sobre ai, i després sobre la llista sat
        bota.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                DatabaseReference refai = FirebaseDatabase.getInstance().getReference("ai");
                DatabaseReference refm = FirebaseDatabase.getInstance().getReference("modat");
                refai.setValue(m);
                refm.child("User1").setValue(m);
            }
        });
        // Exemple de listener de lectura única addValueEventListener()
        // Per a posar el títol. Sobre nomat
        nomat.addValueListener(new ValueEventListener() {
            @Override
            public void onDataChange(DataSnapshot dataSnapshot) {
                String value = dataSnapshot.getValue(String.class);
                ultim.setText(value);
            }
        });
        // Exemple de listener de lectura continua addValueEventListener()
        // Per a posar l'últim missatge registrat. Sobre ai
        refAI.addValueEventListener(new ValueEventListener() {
            @Override
            public void onDataChange(DataSnapshot dataSnapshot) {
                String value = dataSnapshot.getValue(String.class);
                ultim.setText(value);
            }
        });
        // Exemple de listener d'una llista addChildEventListener()
        // Per a posar tota la llista de missatges. Sobre sat
        sat.addValueEventListener(new ChildEventListener() {
            @Override
            public void onChildAdded(DataSnapshot dataSnapshot, String s) {
                String area = dataSnapshot.child("User1").getValue(String.class) + "\n";
                area.append(dataSnapshot.child("User1").getValue(String.class));
                ultim.setText(area);
            }
            @Override
            public void onChildChanged(DataSnapshot dataSnapshot, String s) {
            }
            @Override
            public void onChildRemoved(DataSnapshot dataSnapshot) {
            }
            @Override
            public void onChildMoved(DataSnapshot dataSnapshot, String s) {
            }
            @Override
            public void onCancelled(DatabaseError databaseError) {
            }
        });
    }
}
```

KOTLIN

La classe Message queda molt senzilla:

```
class Message(val nom: String, val contingut: String)
```

I el programa seria aquest:

```
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import com.google.firebase.database.*
import kotlin.android.synthetic.main.activity_main.*

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        bota.text = "Envia"
        // Referències a la Base de Dades i a les variables ai, nomat i sat
        val database = FirebaseDatabase.getInstance()
        val refai = database.getReference("ai")
        val refm = database.getReference("modat")
        val sat = database.getReference("sat")
        // Exemple de guardar dades. Primer sobre ai, i després sobre la llista sat
        bota.setOnClickListener {
            refai.setValue(Message("User1",text.text))
            sat.addValueEvent(object : ValueEventListener {
                override fun onDataChange(snapshot: DataSnapshot) {
                    val value = snapshot.getValue(String::class.java)
                    ultim.text = value
                }
            })
        }
        // Exemple de listener de lectura única addValueEventListener()
        // Per a posar el títol. Sobre nomat
        nomat.addValueEvent(object : ValueEventListener {
            override fun onDataChange(snapshot: DataSnapshot) {
                val value = snapshot.getValue(String::class.java)
                ultim.text = value
            }
        })
        // Exemple de listener de lectura continua addValueEventListener()
        // Per a posar l'últim missatge registrat. Sobre ai
        refAI.addValueEvent(object : ValueEventListener {
            override fun onDataChange(snapshot: DataSnapshot) {
                val value = snapshot.getValue(String::class.java)
                ultim.text = value
            }
        })
        // Exemple de listener d'una llista addChildEventListener()
        // Per a posar tota la llista de missatges. Sobre sat
        sat.addValueEvent(object : ChildEventListener {
            override fun onChildAdded(snapshot: DataSnapshot, key: String) {
                ultim.append("\n")
                ultim.append(snapshot.getValue(String::class.java))
            }
            override fun onChildChanged(snapshot: DataSnapshot, key: String) {
            }
            override fun onChildRemoved(snapshot: DataSnapshot, key: String) {
            }
            override fun onChildMoved(snapshot: DataSnapshot, key: String) {
            }
            override fun onCancelled(error: DatabaseError) {
            }
        })
    }
}
```

```
// Exemple de l'listener d'una llista addChildEventListener()
// Per a posar tota la llista de missatges. Sobre així
val.addValueEventListener(object : ChildEventListener {
    override fun onChildAdded(dataSnapshot: DataSnapshot, s: String?) {
        dataSnapshot.child("nom").getValue(String::class.java) + ":" + dataSnapshot.child("contingut").getValue(String::class.java) + "\n"
    }
    override fun onChildChanged(dataSnapshot: DataSnapshot, s: String?) {
    }
    override fun onChildRemoved(dataSnapshot: DataSnapshot, s: String?) {
    }
    override fun onChildMoved(dataSnapshot: DataSnapshot, s: String?) {
    }
    override fun onCancelled(databaseError: DatabaseError) {
    }
})
```

Tant en el cas de Java com de Kotlin, el resultat seria aquest:



2.3 Cloud Firestore (CF)

Cloud Firestore és l'evolució de Realtime Database, on ja podem guardar més d'un document. Els documents arriban organitzats en col·leccions. Ja podem parlar d'una veritable Base de Dades documental.

Google recomana la utilització de Cloud Firestore en compte de Realtime Database. Potser fins i tot en un futur deixen de proveir aquesta última. Però no seria cap problema perquè tot el que podem guardar en Realtime Database ho podem guardar en Cloud Firestore, sense que se'n complica l'organització de les dades. I l'accés és igual de fàcil.

2.3.1 CF: Utilització des de l'entorn de Firebase

Amb el Cloud Firestore ja podem guardar més d'un document, i aniran agrupats en col·leccions de documents. En el següent vídeo veieu com a inserir en aquest aspecte dels documents, ja que és el més novetós.

Hem de fer el consideració de que encara que trebarem sobre la mateixa Base de Dades de Firebase, des de Cloud Firestore no podem accedir a les dades de Realtime Database, i viceversa.

Aquí té l'estructura de dades que ens hem guardat (per exemple):

Observa també que dins d'un document d'una col·lecció es pot començar una col·lecció i dins d'ella crear documents, etc. De manera que ho podem fer recursiu, i s'entreveuen molt les possibilitats de disseny de la nostra Base de Dades. Anem a aprofundir en aquesta possibilitat de les subcol·leccions per a redissenyar les dades del nostre exemple del xat.

Subcol·leccions

Com es veia en l'última imatge, es poden crear subcol·leccions en un document, i aquelles col·leccions conteran la seua vegada tots els documents que calguin.

Aleshores ens redissenarem el xat comtant en la part de Realtime Database. En aquest moment vam posar tots els missatges del xat en un fitxer. Ens anava molt bé, però després posarem un listener sobre la clau del xat que detectaria tots els elements nous: `el.addValueEventListener(listener)`.

Tenim en el Cloud Firestore no podem el listener anterior. El que si que disposarem és d'un listener que detecti els nous documents sobre una col·lecció. Aleshores anem a muntar cada missatge del xat com un nou document, que de moment només tindrà usat el missatge.

Per tant, després de l'anterior fem una col·lecció anomenada `XatProva`, dins del qual hi ha dos documents, un d'ells amb nom `B` i un altre amb nom general per Cloud Firestore

Exemple

Podeu fer tot el que ve a continuació sobre un programa gràfic:

Aquest és el seu esquema:

```

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Container;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowEvent;
import java.awt.event.WindowListener;
import java.io.IOException;
import java.util.HashMap;
import java.util.Map;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.JTextField;

import com.google.api.core.ApiFuture;
import com.google.auth.oauth2.GoogleCredentials;
import com.google.cloud.firestore.CollectionReference;
import com.google.cloud.firestore.Firestore;
import com.google.cloud.firestore.FirestoreClient;
import com.google.firebase.FirebaseApp;
import com.google.firebase.FirebaseOptions;
import com.google.firebase.auth.FirebaseAuth;
import com.google.firebase.database.ChildEventListener;
import com.google.firebase.database.DataSnapshot;
import com.google.firebase.database.DatabaseError;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;
import com.google.firebase.database.ValueEventListener;
import com.google.firebase.storage.StorageReference;
import com.google.firebase.storage.UploadTask;

public class Pantalla_CrearKatCloud extends JFrame implements ActionListener {

    private static final long serialVersionUID = 1L;
    JLabel ultimmissatge = new JLabel("Últim missatge: ");
    JLabel ultimmissatge2 = new JLabel();
    JLabel etiqueta = new JLabel("Missatge:");
    JTextArea area = new JTextArea();
    JLabel introduixmissatge = new JLabel("Introdueix missatge:");
    JTextField missatge = new JTextField("Scriu");
    JTextField missatge2 = new JTextField();

    // en iniciar posem un contenedor per als elements anteriors
    public void iniciar() throws IOException {
        JFrame pantfPrincipal = this;
        this.setSize(300, 300);
        this.setLayout(new BorderLayout());
        // contenedor per als elements
        JPanel panel1 = new JPanel(new GridLayout());
        panel1.add(ultimmissatge);
        panel1.add(ultimmissatge2);
        getcontentPane().add(panel1, BorderLayout.NORTH);

        JPanel panel2 = new JPanel(new BorderLayout());
        panel2.add(etiqueta, BorderLayout.CENTER);
        JScrollPane scroll = new JScrollPane(area);
        panel2.add(scroll, BorderLayout.CENTER);
        getcontentPane().add(panel2, BorderLayout.CENTER);

        JPanel panel3 = new JPanel(new GridLayout());
        panel3.add(introduixmissatge);
        panel3.add(missatge);
        panel3.add(missatge2);
        getcontentPane().add(panel3, BorderLayout.SOUTH);

        setVisible(true);
        enviar.addActionListener(this);
    }

    @Override

    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == enviar) {
            // Exemple de guardar dades sense haver d'esperar per ser una aplicació gràfica
            // Per a posar el títol. Sobre Xati.Xati.memoCat
            // Exemple de llistar i d'eliminar dades sense haver d'esperar per ser una aplicació gràfica
            // Per a posar l'últim missatge registrat. Sobre Xati.Xati.al
            // Exemple de llistar i d'eliminar dades sense haver d'esperar per ser una aplicació gràfica
            // Per a posar tota la llista de missatges. Sobre Xati.Xati.cat
        }
    }

    I aquest és el programa principal que crida l'anterior
}

import java.io.IOException;

public class CrearKatCloud {
    public static void main(String[] args) throws IOException {
        Pantalla_CrearKatCloud finestra = new Pantalla_CrearKatCloud();
        finestra.setVisible(true);
    }
}

```

2.3.2.1 CF-Java: Connexió des de Java

Drivers necessaris

Els drivers necessaris són els mateixos que van baixar-nos per al cas de **Realtime Database**. Millor dit, estan inclosos en la llibreria que ens vam muntar, per tant aquesta feina la vam fer en el punt **2.2.2.1**.

Configuració

Exactament el mateix que en el cas de **Realtime Database**. Observeu que és lògic, perquè és la manera d'accorder al projecte, i el projecte és el mateix. Per tant, tornem a fer referència al punt **2.2.2.1**.

Recordeu que ens vam baixar un **firebase.json** amb la clau privada que vam guardar a l'arxiu del projecte (i del qual és molt convenient guardar còpia). Després, com deia l'exemple, col loquem el següent per a un accés correcte:

```
FileInputStream serviceAccount = new FileInputStream("moe_a1_firebase.json");
FirebaseOptions options = new FirebaseOptions.Builder()
.setCredentials(GoogleCredentials.fromStream(serviceAccount))
.setDatabaseUrl("https://acces-a-dades-firebase.firebaseio.com").build();
FirebaseApp.initializeApp(options);
```

No us oblideu de substituir el nom del **firebase.json**. Tant bé ho tenir en compte que la URL o la base de dades serà diferent per a cada cas de nosaltres.

Ara serà lleugerament més que en el cas de **Realtime Database** visto en el punt **2.2.2.1**, ja que no caldrà especificar la URL de la Base de Dades, amb la referència al **firebase.json** serà suficient.

Recordeu que ens vam baixar un **firebase.json** amb la clau privada que vam guardar a l'arxiu del projecte (i del qual és molt convenient guardar còpia). Després, com deia l'exemple, col loquem el següent per a un accés correcte, on no posem el **.setDatabaseURL()**:

```
FileInputStream serviceAccount = new FileInputStream("moe_a1_firebase.json");
FirebaseOptions options = new FirebaseOptions.Builder()
.setCredentials(GoogleCredentials.fromStream(serviceAccount))
.setDatabaseURL(null);
FirebaseApp.initializeApp(options);
```

No us oblideu de substituir el nom del **firebase.json**.

Referència a la Base de Dades i a les dades concretes a les quals volem accedir

Ara la referència a la Base de Dades de Cloud **Firebase** canviarà:

```
Firestore database = FirestoreClient.getFirestore();
```

Com que la informació està organitzada en col·leccions i dins d'ella en documents, podem accedir primer a les col·leccions i dins d'elles a documents. A continuació tens la referència pas a pas:

```
CollectionReference col = database.collection("Xats");
DocumentReference docXatProva = col.document("XatProva");
```

Ensara que evidentment ho podem haver fet en una única línia:

```
DocumentReference docXatProva = database.collection("Xats").document("XatProva");
```

No ens plantejarem accés a un element de dins d'un document, sinó que de moment accedirem a tot el conjunt del document.

Recordeu que **Realtime Database**, com era un únic document JSON, alla aranya tot i obligatoriament havíem de poder tenir una referència a un únic element. Com en **Cloud Firestore** ho podem organitzar en col·leccions i documents, cadaçun d'ells si poden dissenyar millor i que continga únicament les dades estríctes. Aleshores ens anirà bé accedir a tot el document.

Guardar dades

Com acaben de comentar, accedim a tot un document direcament.

Per a guardar dades, ens podem plantejar 3 operacions d'escriptura sobre el document:

- Subscriure's (`set`) ho farem amb el mètode `set()` com el mètode `update()` accepten com a paràmetre no una única dada, sinó una estructura que puga aniar a reflectir el document. Acceptaran com a paràmetre un `Map<String, Object>`, on podrem col·locar les claus i els valors de tots els membres del document. Cada valor pot ser dels tipus que vam practicar en el punt anterior: string, nombre, boolean, array, map.
- Eliminar-lo tot: amb el mètode `delete()`
- Modificar-lo: amb el mètode `update()`

Excepte per editar, per a les altres operacions ens fa falta saber l'estructura del document. Per això (tant mètode `set()` com el mètode `update()`) accepten com a paràmetre no una única dada, sinó una estructura que puga aniar a reflectir el document. Acceptaran com a paràmetre un `Map<String, Object>`, on podrem col·locar les claus i els valors de tots els membres del document. Cada valor pot ser dels tipus que vam practicar en el punt anterior: string, nombre, boolean, array, map.

La manxa de col·locar un element en una estructura `Map<String, Object>` es farà amb el mètode `put()`, que acceptarà dos paràmetres: la clau i el valor.

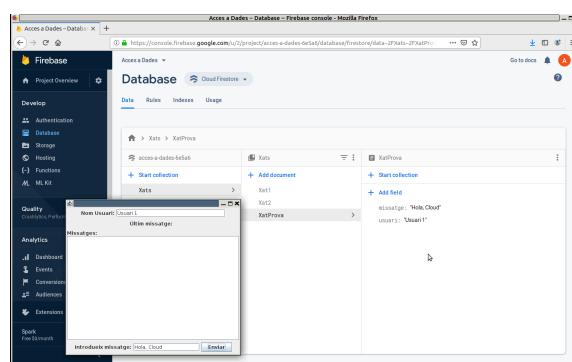
En el següent exemple, quan s'aplica el botó Envíar (baix a la dreta) es guardarà el contingut del quadre de text `usuari` que està dins i el de `missatge` que està baix. Però observeu que estem utilitzant el mètode `set()` i per tant malbaratarem el que ja tenim construït (el nom del sat i la subcolecció de missatges). **Per tant US ACONSEGELLA QUE NO EXECUTEU EL QUE VE A CONTINUACIÓ**

```
public void actionPerformed(ActionEvent e) {
    if (e.getSource() == enviar) {
        // Exemple de guardar dades en Cloud Firestore
        // Per a guardar dades. Sobre /Xats/XatProva i després sobre /Xats/Xat1
        Firestore database = FirestoreClient.getFirestore();
        DocumentReference docXatProva = database.collection("Xats").document("XatProva");

        Map<String, Object> dades = new HashMap<>();
        dades.put("ultimoUser", usuari.getText());
        dades.put("ultimoMissatge", missatge.getText());

        docXatProva.set(dades);
    }
}
```

Aquest seria el resultat, creant-se el document amb les 2 clau-valor, i havent malbaratat el que hi havia



En realitat els mètodes `set()`, `delete()` i `update()` tornen un valor, que és un `ApiFuture<WriteResult>`, que permet averguer com ha anat l'actualització de les dades. D'aquest `ApiFuture<...>` podem obtenir el moment en què s'ha confirmat l'actualització, però trancant-lo perquè pot donar error.

Ampliem l'exemple anterior en què únicament usarem el `set()` per a comprovar el que comentem, senzillament mostrant el moment en què s'ha actualitzat. **US ACONSEGELLA QUE NO EXECUTEU EL QUE VE A CONTINUACIÓ**

```
public void actionPerformed(ActionEvent e) {
    if (e.getSource() == enviar) {
        // Exemple de guardar dades en Cloud Firestore
        // Per a guardar dades. Sobre /Xats/XatProva i després sobre /Xats/Xat1
        Firestore database = FirestoreClient.getFirestore();
        DocumentReference docXatProva = database.collection("Xats").document("XatProva");

        Map<String, Object> dades = new HashMap<>();
        dades.put("ultimoUser", usuari.getText());
        dades.put("ultimoMissatge", missatge.getText());

        ApiFuture<WriteResult> result = docXatProva.set(dades);
        try {
            System.out.println("Time : " + result.getUpdateTime());
        } catch (InterruptedException e1) {
            e1.printStackTrace();
        } catch (ExecutionException e1) {
            e1.printStackTrace();
        }
        e1.printStackTrace();
    }
}
```

Si veiem la consola d'Eclipse veurem en negre al final com ha imprès el moment de la confirmació.

```
[14:42:44] Failed to load class [org.eclipse.jdt.launching.ILaunch]
[14:42:44] Getting the class from the JRE. This might be a bug in the JRE.
[14:42:44] See https://bugs.eclipse.org/bugs/show_bug.cgi?id=44000 for further details.
[14:42:44] WARNING: The behavior for java.util.Date objects stored in Firestore is going to change AND YOUR APP MAY BREAK.
[14:42:44] In the past, Java.util.Date objects stored in Firestore were automatically converted to UTC timestamps when reading from the database. To avoid data loss, we now convert them back to local dates using the device's clock when reading from the database.
[14:42:44] Please add or update existing usages of java.util.Date when you enable the new behavior. In a future release, the behavior will be changed to the new behavior, so if you do not
update your code before then, your app may break.
[14:42:44] update time : 2018-12-10T12:31:09.151Z+0000
```

Apertençom per comentar els usos que ens han arribat. A les 4 primeres línies no els donarem importància. A partir de la quarta ens avisa fins a quina cosa que pràcticament podria causar que fallaria el programa, i és el tractament de les dades. En el nostre exemple no cal que ens preocupem, perquè no guardem dates, i en els exercicis les guardem com un `long`.

En els exemples anteriors, en cas d'haver-se executat manualment el document `XatProva` (i no per haver utilitzat el mètode `set()`).

Ara començaré el mètode `update()`, en què senzillament apliquem els dades que proporcionem, però mantenint els altres. **AQUEST EXEMPLER SÍ QUE EL PODEU PER**

```
public void actionPerformed(ActionEvent e) {
    if (e.getSource() == enviar) {
        // Exemple de guardar dades en Cloud Firestore
        // Per a guardar dades. Sobre /Xats/XatProva i després sobre /Xats/Xat1
        Firestore database = FirestoreClient.getFirestore();
        DocumentReference docXatProva = database.collection("Xats").document("XatProva");

        Map<String, Object> dades = new HashMap<>();
        dades.put("ultimoUser", usuari.getText());
        dades.put("ultimoMissatge", missatge.getText());

        docXatProva.update(dades);
    }
}
```

conservarem tant el camp `nomXat` com la col·lecció `messages`

Recuperar dades

Com en el cas del Realtime Database, ens plantejaren dos casos:

- Una única lectura
- Un listener que es quede escuchant per si hi ha canvis

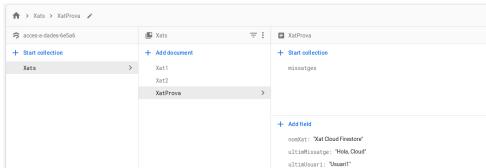
Com comprarem, queda més senzilla la lectura única en el cas de Cloud Firestore.

Lectura única

Per a fer una lectura únicament utilitzarem el mètode `get()` sobre una referència al document que volem llegir. El resultat ens ve en un `ApiFuture<DocumentSnapshot>`, que com es veu és una còpia de les dades en una interface `ApiFuture`, indicant que la tindrem disponible en un futur. Sobre l'anterior farem `get()` per a obtenir aquesta còpia (serà un objecte). I sobre ella podrém fer:

- `getDocument()` per obtenir tot el document en un `Map<String, Object>`
- `getSnapshot()` per a obtenir tot el document en un `DocumentSnapshot`
- `getDocumentCount()` per a obtenir directament el valor d'un del document
- `getString(String)` per a obtenir el valor de la clau en forma de string
- `getDouble(String)` per a obtenir el valor de la clau en forma de double

* L'exemple d'únic lectura el farem per a posar el títol de l'aplicació, com en l'exemple del Realtime Database. Prèviament, des de la consola de Firebase ja havíem creat la parella clau-valor: `nomXat: Xat Cloud Firestore`, dins del document `XatProva`. Si els exemples anteriors (els que havíem marcat en roig que no us aconseguíen fer) heu perdut aquella parella, és el moment de crear-la de nou:



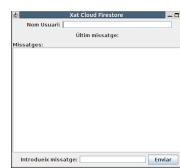
Ara si que hauríem d'anar amb compte en el moment de guardar les dades. Si utilitzem el mètode `set()` malbaratarem el document anterior (amb el nom del sat). Per tant, a partir d'això ens convindrà utilitzar el mètode `update()`. Ho tornarem a especificar en el moment de fer aquesta escriptura.

I aquestes seran les semàntiques per a posar el títol de l'aplicació:

```
// Exemple de lectura única: consultant sobre un ApiFuture i sobre ell get()
// Per a posar el títol. Sobre /Xats/XatProva/nomXat
Firestore database = FirestoreClient.getFirestore();
DocumentReference docXatProva = database.collection("Xats").document("XatProva");
ApiFuture<DocumentSnapshot> future = docXatProva.get();
```

```
String nomXat = future.get().getString("nomXat");
this.setTitle(nomXat);
```

I així tenim el resultat:



Listener que es queda escoltant: addSnapshotListener()

De forma paralela al Realtime Database, si volem rebre una notificació de quan un canvi en el document que ens interessa, sofití una referència a aquest document ens muntarem un listener, en aquest cas amb el mètode `addSnapshotListener()`. Al mètode `onEvent()` que s'ha de sobreescrivir arribarà un parametre de tipus `DocumentSnapshot`, que serà una còpia del document. Com en el cas anterior podem fer sobre el un `get(Data)` per a obtenir tot el document, `getString(FieldPath)` per a obtenir el valor de la clau com a string, etc.

En el nostre exemple, de moment utilitzem tant per a posar un missatge com per a anar omplint l'àrea central amb el xat. Però com havíem comentat abans, hem de cuidar de no rebaixar tot el document per a noo perdre el títol del xat. Per tant, fàcio que faríem en apretar el botó per a enviar el missatge no serà `set()` sinó `update()`. Per si de cas havíeu fet els exemples que us havíem aconsellat que no fereu, així teniu una altra vegada el codi correcte:

```
public void actionPerformed(ActionEvent e) {
    if (e.getSource() == enviar) {
        // Exemple de guardar dades en Cloud Firestore
        // Per a guardar dades. Sobre /Xats/XatProva/1 sebordes sobre /Xats/Xat
        FirebaseFirestore database = FirebaseFirestore.getInstance();
        DocumentReference docXatProva = database.collection("Xats").document("XatProva");

        Map<String, Object> dades = new HashMap<>();
        dades.put("ultimoUser", user1.getText());
        dades.put("ultimoMessage", message.getText());

        docXatProva.update(dades);
    }
}

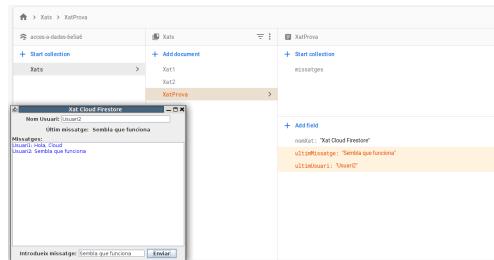
I ara si el addSnapshotListener():

// Exemple de listener de lectura continua addSnapshotListener()
// Per a posar l'últim missatge registrat. Sobre /Xats/XatProva/ultimoUser i /Xats/XatProva/ultimoMessage
database.addSnapshotListener(new EventListener<DocumentSnapshot>() {
    @Override
    public void onEvent(@Nullable DocumentSnapshot snapshot,
                        @Nullable FirebaseFirestoreException e) {
        if (e != null) {
            System.err.println("Listen failed: " + e);
            return;
        }

        if (snapshot != null && snapshot.exists()) {
            ultimoMessage.setText(snapshot.getString("ultimoMessage"));
            area.append(snapshot.getString("ultimoUser") + ":" + snapshot.getString("ultimoMessage") + "\n");
        } else {
            System.out.print("Current data: null");
        }
    }
});
```

En aquest exemple tenim el tractament de possibles errors, com que no s'accedeix al document, o aquest és nul.

Observeu com la primera lectura també la fa, del que hi ha ja guardat en principi. En la imatge es mostra el moment d'afegir un segon missatge:



Guardar documents

Ja havíem comentat en la pregunta 2.3.1 que per a l'estrucció de les dades a la qual ens compta Cloud Firestore, en compte de guardar els missatges (i l'usuari) en una llista, ho faríem en documents direts d'una subcoll col·lecció.

Per tant ens és necessària l'operació d'afegir un document a una col·lecció. Avui en un principi ho aconseguirem amb el mètode `set()` sobre un document nou de la col·lecció:

```
database.collection("nomCol").document("nomDoc").set(dades);

però això ens obligaria a posar un nom a cada document. Ja havíem vist que Cloud Firestore era capaç de generar un nom de document que no es puga repetir. Des de Java s'aconsegueix amb el mètode add():

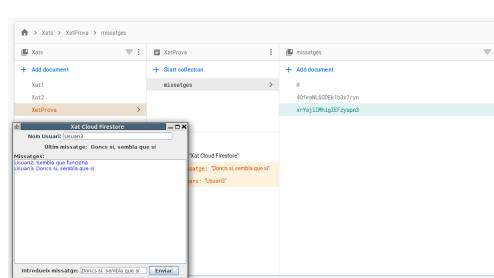
database.collection("nomCol").add(dades);

però que en el nostre cas, per ser una subcoll col·lecció és un poc més llarg

database.collection("Xats").document("XatProva").collection("missatges").add(dades);

L'estrucció de les dades la podem fer amb un Map<String, Object>>String, Object>: i posarà en el nostre cas l'usuari i el missatge. D'aquesta manera ens quedarà així el procediment en apretar el botó d'enviar el missatge, on a banda del que tenim abans per a modificar ultimoMessage i ultimoUser, així alegrem el document nou.
```

I aquest és el resultat:



On podem veure com el contingut del nou document és el que volem:

The screenshot shows the Firebase Realtime Database interface. A collection named 'missatges' is selected. Inside, there is a single document with the ID '49fca622d1b5c7c9'. The document's data is as follows:

```

{
  "missatges": {
    "49fca622d1b5c7c9": {
      "usuari": "UserCloudFire",
      "text": "Dona's el sentit que hi ha",
      "ultimMessage": "Dona's el sentit que hi ha",
      "ultimUser": "UserCF"
    }
  }
}

```

Recuperar documents modificats

Ja només ens queda detectar els canvis en els documents de la col·lecció, per a afegeir a l'assa central els documents afegits. També és un addSnapshotListener(), però ara l'apliquem a una col·lecció (no a un document). El resultat és que d'una forma molt comoda podem detectar els documents afegits, els modificats i fins i tot els elaborats.

Així tenim el fragment de programa que ens ho permetrà. Hem deixat els casos de document modificat i document elaborat per a una millor documentació, encara que en el nostre exemple no ho utilitzem.

```

// Exemple de llistador de lectura continua addSnapshotListener() sobre una col·lecció
// Per a posar tota la llista de missatges. Sobre "/XatPrva/missatges"
database.collection("XatPrva").collection("missatges").addSnapshotListener(new EventListener<QuerySnapshot>() {
    @Override
    public void onEvent(@Nullable QuerySnapshot snapshots,
                        @Nullable FirebaseFirestoreException e) {
        if (e != null) {
            System.out.println("Error: " + e);
            return;
        }

        for (DocumentChange dc : snapshots.getDocumentChanges()) {
            switch (dc.getType()) {
                case ADDED:
                    System.out.println(dc.getDocument().getString("usuari") + ": " + dc.getDocument().getString("missatge") + "\n");
                    break;
                case MODIFIED:
                    System.out.println("Missatge modificat: " + dc.getDocument().getData());
                    break;
                case REMOVED:
                    System.out.println("Missatge esborrat: " + dc.getDocument().getData());
                    break;
                default:
                    break;
            }
        }
    }
});

```

I en aquesta imatge es veu com en un principi es veuen els 3 documents que ja existien (2 creats en la pregunta 2.3.1, i 1 immediatament abans). També s'afegeix un altre missatge per veure que es visualitza perfectament.



Es de destacar que ara la còpia de les dades, el snapshot, és en realitat un **QuerySnapshot**. I és que nosaltres no hem fet cap consulta sobre els documents de la col·lecció, però és possible fer-la. Per exemple es podria seleccionar per mig d'una query tots els documents en què l'usuari és un determinat. Així doncs detectarem els documents afegits, modificats o esborrats d'aquest usuari. Ho farem en el moment de declarar el **addSnapshotListener()**.

```
database.collection("XatPrva").document("UserPrva").collection("missatges").whereEqualTo("usuari", "UserPrva").addSnapshotListener(new EventListener<QuerySnapshot>() {
```

O sembla podríem ordenar els documents per algun camp. Això en realitat seria necessari per a poder ordenar els missatges, ja que el nom del document auto-generat per Cloud Firestore no és consecutiu, sinó aleatori, i per tant perfectament un document nou tinga un nom anterior a alguns dels existents. Ho arreglarem fàcilment posant un camp més amb la data i ordenant per aquest camp amb **orderBy()** (que arriba en el final del **whereEqualTo()**) de la sentència anterior.

Així es pot fer l'exemple del tel complet, amb el que algunes modificacions i milles:

- Treuen la classe Message amb els propietats nom i contingut (expressament diferents dels noms utilitzats en l'exemple anterior per poder comprovar que aquestes modificacions funcionen i milloren l'exemple). Posarem també la data (com un long) per a poder ordenar els missatges cronològicament
- Lieven les coses que no s'utilitzen explicitament en aquest exemple
- No definim la referència a la Base de Dades en cada úsitz. Únicament una vegada al principi, després de la connexió.
- Posarem un comentari per a poder fer entre-més d'un sat. Per a no modificar les coses ja fets, posarem la creació dels listeners en un mètode anomenat `inicialitzar()`, ja que ara els listeners depenen del tel that. I farem d'afair amb compte fàcil de parlar els listeners, ja creus. Es a dir, si primer them un sat, tindrem els listeners que apuntin a ell. Si després them un altre sat i creem els listeners una altra vegada per a que apunten al nou

Aquest és la classe Message

```
public class Message{
    public String nom;
    public long data;
    public String contingut;
    public Message(){
    }
    public Message(String nom, String contingut){
        this.nom = nom;
        this.contingut = contingut;
    }
    public Message(String nom, long data, String contingut) {
        super();
        this.nom = nom;
        this.data = data;
        this.contingut = contingut;
    }
    public String getNom() {
        return nom;
    }
    public void setNom(String nom) {
        this.nom = nom;
    }
    public long getData() {
        return data;
    }
    public void setData(long data) {
        this.data = data;
    }
    public String getContingut() {
        return contingut;
    }
    public void setContingut(String contingut) {
        this.contingut = contingut;
    }
}
```

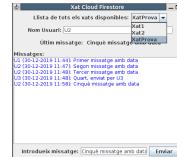
Aquest és el programa

```
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.FlowLayout;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowEvent;
import java.io.IOException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.concurrent.ExecutionException;
import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.JTextField;
import javax.swing.JLabel;
import com.google.api.core.ApiFuture;
import com.google.auth.oauth2.GoogleCredentials;
import com.google.cloud.firestore.DocumentChange;
import com.google.cloud.firestore.DocumentSnapshot;
import com.google.cloud.firestore.EventListener;
import com.google.cloud.firestore.FirestoreException;
import com.google.cloud.firestore.ListenerRegistration;
import com.google.cloud.firestore.QueryDocumentSnapshot;
import com.google.cloud.firestore.QuerySnapshot;
import com.google.firebase.FirebaseApp;
import com.google.firebase.cloud.FirestoreClient;
import com.google.firebase.database.annotations.Nullable;
public class Pantalla_CrearCloud extends JFrame implements ActionListener {
    private static final long serialVersionUID = 1L;
    JLabel etxComCombobox = new JLabel("Lista de tots els xats disponibles:");
    JComboBox comboBoxXat = new JComboBox();
    JLabel etxUser = new JLabel("Nom Usuari:");
    JTextField user = new JTextField(20);
    JLabel etxUltmissatge = new JLabel("Últim missatge:");
    JLabel ultimmissatge = new JLabel();
    JLabel etiqueta = new JLabel("Missatge:");
    JTextArea area = new JTextArea();
    JLabel etxIntroducirmissatge = new JLabel("Introdueix missatge:");
    JButton enviar = new JButton("Enviar");
    JTextField missatge = new JTextField(10);
    Firestore database = null;
    DocumentReference docRef = null;
    ListenerRegistration listenerUltmissatge = null;
    ListenerRegistration listenerMissatges = null;
    // en iniciar posem un contenidor per als elements anteriors
    public void iniciar() throws IOException, InterruptedException, ExecutionException {
        JFrame pantxaPrincipal = this;
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setSize(400, 400);
        this.setLayout(new BorderLayout());
        // contenidor per als elements
        // hi haurà titol, Panel de dalt: missatge, Panel de baix per a introduir missatge. Panel central: tot el sat
        JPanel panel10 = new JPanel(new FlowLayout());
        panel10.add(etxComCombobox);
        panel10.add(comboBoxXat);
        JPanel panel11 = new JPanel(new FlowLayout());
        panel11.add(etxUser);
        panel11.add(user);
        JPanel panel12 = new JPanel(new FlowLayout());
        panel12.add(etxUltmissatge);
        panel12.add(ultimmissatge);
        JPanel panel13 = new JPanel(new GridLayout(3, 1));
        panel13.add(area);
        panel13.add(etiqueta);
        panel13.add(missatge);
        panel13.add(enviar);
        getContentsPane().add(panel10, BorderLayout.NORTH);
        JPanel panel14 = new JPanel(new BorderLayout());
        panel14.setLayout(new BorderLayout());
        area.setEditable(true);
        area.setBackground(Color.blue);
        JScrollPane scroll = new JScrollPane(area);
        panel14.add(scroll, "Center");
        getContentsPane().add(panel14, BorderLayout.CENTER);
        JPanel panel15 = new JPanel(new FlowLayout());
        panel15.add(etxIntroducirmissatge);
        panel15.add(enviar);
        getContentsPane().add(panel15, BorderLayout.SOUTH);
        service();
        comboboxData.addActionListener(this);
        enviar.addActionListener(this);
        FileInputStream serviceAccount = new FileInputStream(
            "acess-a-dades-de-datal-firebase-adminsdk-e1fcf-8f5d926921.json");
        GoogleCredentials credentials = GoogleCredentials.fromStream(serviceAccount);
        FirebaseOptions options = new FirebaseOptions.Builder().setCredentials(credentials).build();
        FirebaseFirestore.initializeApp(options);
        database = FirestoreClient.getFirestore();
        // Exemple de llegir tots els documents d'una col·lecció
        // Per a triar el xat
        List<QueryDocumentSnapshot> documents = database.collection("xats").get().getDocuments();
        for (QueryDocumentSnapshot document : documents) {
            ComboBoxXat.addItem(document.getId());
        }
        ComboBoxXat.setSelectedIndex(0);
    }
    private void inicialitzar() throws InterruptedException, ExecutionException {
        // Tindràs auto-generated method stub
        docRef = database.collection("xats").document(ComboBoxXat.getSelectedItem().toString());
        area.setText("");
    }
    // Exemple de lectura única: senzillament sobre un Apifuture i sobre ell get()
    // Per a posar el títol. Sobre /xats/xat/porcavonsat
    Apifuture<String> futureGetXat();
    String nomeXat = futureGetXat().get();
    this.setTitle(nomeXat);
}
```

```
I aquesta és el programa principal

import java.io.IOException;
import java.util.concurrent.ExecutionException;

public class CrearXatCloud {
    public static void main(String[] args) throws IOException, InterruptedException {
        final PantallaCrearXatCloud finestra = new PantallaCrearXatCloud();
        finestra.iniciar();
    }
}
```



2.3.3 CF: Utilització des d'Android

De forma absolutament paral·lela a com quan vam veure el Realtime Database per a Android, ara tots el torn d'accés al Cloud Firestore des d'Android. Com veurem, per a fer la connexió ens deixarem ajular també per l'assistent.

Com en el cas anterior, farem dues versions, per a Java i per a Kotlin. Aquestes dues versions tenen la mateixa paraula, és a dir el `activity_main.xml` serà identic.

Els projectes s'anomenaran respectivament `Tema7_1_FirebaseCF_Java` i `Tema7_1_FirebaseCF_Kotlin`.

Nota important
 Tant en el cas de JAVA com en el de KOTLIN, per a accedir a Cloud Firestore i que no ens fail el programa, haurem de posar en el `build.gradle` de l'aplicació, dins del `android / dms / defaultConfig` el següent: `multidexEnabled true`

Si no ho hem fet ens pot donar el següent error:
`Cannot find requested classes in a single dex file (F methods: 80444 > 65536)`

```
<activity_main.xml>
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools" tools:context=".MainActivity">
    <androidx.constraintlayout.widget.ConstraintLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        tools:context=".MainActivity">
        <Spinner
            android:id="@+id/combatats"
            android:layout_width="30dp"
            android:layout_height="30dp"
            android:layout_margin="10dp"
            android:layout_centerInParent="true"/>
        <EditText
            android:id="@+id/nomUser"
            android:layout_width="30dp"
            android:layout_height="wrap_content"
            android:layout_margin="10dp"/>
        <app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
        <TextView
            android:id="@+id/resultado"
            android:layout_width="30dp"
            android:layout_height="wrap_content"
            android:layout_margin="10dp"/>
        <app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="parent" />
        <EditText
            android:id="@+id/texto"
            android:layout_width="313dp"
            android:layout_height="44dp"
            android:layout_margin="10dp"/>
        <app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="parent" />
        <Button
            android:id="@+id/bota"
            android:layout_width="wrap_content"
            android:layout_height="44dp"
            android:layout_margin="10dp"/>
        <app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent" />
        <TextView
            android:id="@+id/area"
            android:layout_width="30dp"
            android:layout_height="30dp"
            android:layout_margin="10dp"/>
        <app:layout_constraintBottom_toTopOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="parent" />
    </androidx.constraintlayout.widget.ConstraintLayout>
```

Els components que hem fet són els següents:

- Un `EditText` per al nom d'usuari
- Un `TextView` anomenat `resultado`, per a visualitzar l'últim missatge, que es coordinat amb `Xats/XatProvultimeMessage`
- Un `EditText` anomenat `texto`, per a posar els missatges que volem enviar
- Un `Button` anomenat `bota`, que serveix per enviar el missatge
- Un `TextView` anomenat `area`, on posa el resultat
- Un `Spinner`, per a poder seleccionar el xat entre els quatre existents

I aquest serà l'"esquema" del program:

JAVA

```
import android.app.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;
import android.content.Context;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Spinner;
import android.widget.TextView;
import com.google.android.gms.tasks.OnCompleteListener;
import com.google.android.gms.tasks.OnSuccessListener;
import com.google.android.gms.tasks.Task;
import com.google.firebase.DocumentChange;
import com.google.firebase.firestore.DocumentSnapshot;
import com.google.firebase.firestore.FirebaseFirestore;
import com.google.firebase.firestore.QuerySnapshot;
import com.google.firebase.firestore.WriteResult;
import com.google.firebase.firestore.FirebaseFirestoreException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;
import java.util.List;
import java.util.annotation.Nullable;
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        setContentViewById(R.layout.activity_main);
        final Spinner combatats = (Spinner) findViewById(R.id.combatats);
        final TextView resultado = (TextView) findViewById(R.id.resultado);
        final TextView area = (TextView) findViewById(R.id.area);
        final EditText texto = (EditText) findViewById(R.id.texto);
        final Button bota = (Button) findViewById(R.id.bota);
        bota.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                resultado.setText("");
                Context contextPrincipal = this;
                // Referències a la Base de Dades i als documents
                // Exemple de llegir tots els documents d'una col·lecció
                // Per a triar el xat
                // Exemple de lectura única. AddOnSuccessListener()
                // Per a posar el títol. Sobre Xats/XatProvultimeMessage
                // Exemple de llistar de lectura continua addSnapshotListener() sobre una col·lecció
                // Per a posar l'últim missatge registrat. Sobre Xats/XatProvultimeMessage
                // Exemple de llistar de lectura continua addSnapshotListener() sobre una col·lecció
                // Per a posar tota la llista de missatges. Sobre Xats/XatProvmissatges
            }
        });
    }
}
```

KOTLIN

```
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import com.google.firebase.firestore.*
import java.text.SimpleDateFormat
import java.util.*

import kotlin.android.synthetic.main.activity_main.*

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        dato.text = "Eva"
        val contextPrincipal = this
        // Referències a la Base de Dades i als documents
        // Exemple de llegir tots els documents d'una col·lecció
        // Per a triar el xat
        // Exemple de lectura única. AddOnSuccessListener()
        // Per a posar el títol. Sobre Xats/XatProvultimeMessage
        // Exemple de llistar de lectura continua addSnapshotListener() sobre un document
        // Per a posar l'últim missatge registrat. Sobre Xats/XatProvultimeMessage
        // Exemple de llistar de lectura continua addSnapshotListener() sobre una col·lecció
        // Per a posar tota la llista de missatges. Sobre Xats/XatProvmissatges
    }
}
```

```
// Per a guardar dades
// Primer sobre /xats/XatProva/ultimUserari i /xats/XatProva/ultimMissatge
// Després tindrà com a documents en la col·lecció /xats/XatProva/missatges
}
```

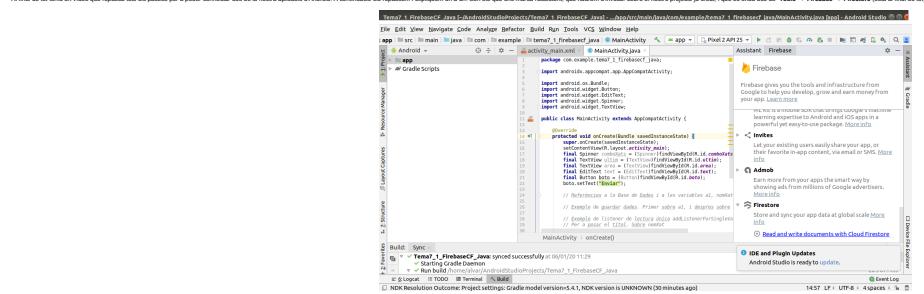
Bases de Dades

http://localhost:51235/temp_print_dirs/eXeTemp...

L'accés és extraordinàriament fàcіt gràcies als assistents que ens proporciona el proje Android Studio.

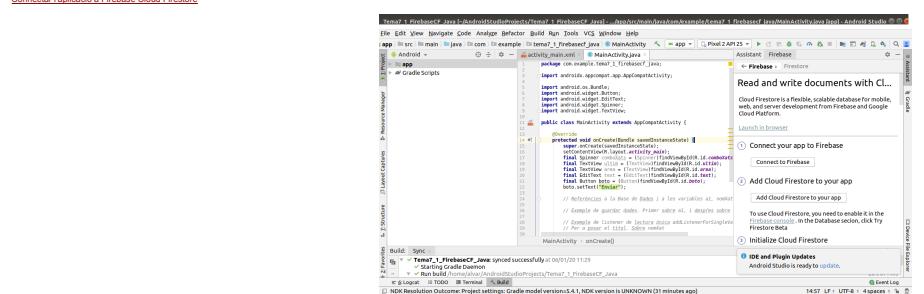
A final de tot tenim un vídeo que repasse tots els passos per a poder connectar des de la nostra aplicació d'Android. A continuació els repassarem i expliquem un a un. Són els que ens marquen l'assistent, que haurem d'iniciar sobre el nostre projecte ja creat. I que es crida des de Tools -> Firebase -> Firestore (esta al final del tot).

2.3.3.1 CF-Android: Connexió



Observem que quasi tot és idèntic, encara que utilitzem Java o Kotlin. Només farem la diferenciació en l'exemple, on està ja tot el codi.

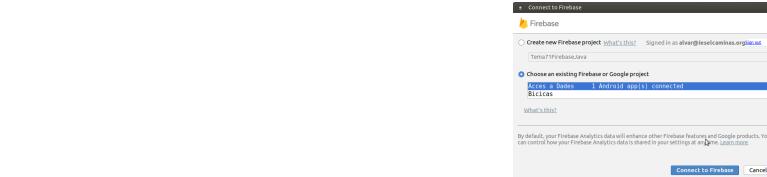
Connectar l'aplicació a Firebase Cloud Firestore



En apretar el botó de **Connect to Firebase**, si no estàvem connectats amb el compte de Google al Firebase se'n obrirà finestra d'un navegador per a conectar. Podrà donar-se el cas que en diguem que Android Studio vol accedir a les dades de la Base de Dades. Obviament ho haurem de permetre:



Una vegada autenticats en Firebase, des de l'entorn d'Android Studio ens ofereix la possibilitat de crear una aplicació nova (una Base de Dades nova) o utilitzar alguna de les ja tems. Utilitzarem la que ens ha servit de prova fins el moment:



Quan heu connectat substituir el botó **Connect to Firebase**, per una etiqueta que dirà **connected**, en verd.

Afegeix la Base de Dades a la nostra aplicació

En aquest segon pas, quan aprem el botó **Add Cloud Firestore to your app**, ens dirà els canvis que farà per a incorporar les coses necessàries per a poder connectar.



Com veieu es tracta d'incloure les llibreries necessàries de Firebase. En el vídeo del final es mostren imatges amb les dades incorporades.

Igual que abans, substituirem el botó **Add the Realtime Database to your app**, per una etiqueta que dirà **Dependencies set up correctly**, en verd. Es una bona guia per saber en quin punt estem.

Permetre l'accés als usuaris, si es prenigant les regles d'accés a la Base de Dades

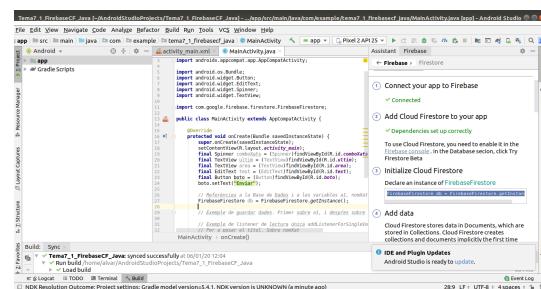
Com que Runtime Database i Cloud Firestore, encara que sembla que estan compartint la Base de Dades, en realitat no es pot accedir d'una i l'altra, igual ens passarà amb les regles d'accés (Rules). Per tant és convenient pagar-hi una mirada. Recordeu que es dàs de la consola de Firebase, entrant al Cloud Firestore, i anant a les pàgines Rules.



Observem que en aquest exemple he posat que es pot connectar qualsevol dia al 30-12-2020. Inicialment dóna un mes, però si no voleu fer-ho, podeu allargar canviant la data.

Copiar les sentències per a escriure i per a llegir (míllor dit per a detectar els canvis en temps real)

Era diu un exemple de les sentències a copiar per a poder fer referència a la Base de Dades, per a guardar una informació a la Base de Dades i també per a detectar un canvi en la Base de Dades i poder obtenir el seu valor. Encara que tinguis un projecte en Kotlin, les sentències d'exemple seran de Java, però que en copiar-les les tradusira a Kotlin. No fem diferenciació per tant entre Java i Kotlin en aquest moment.



Referència a la Base de Dades i a les dades concretes a les quals volem accedir.

Podrem fer referència, a banda de la Bases de Dades, a cada col·lecció i/o a cada document de cada col·lecció

JAVA

```
// Referències a la Base de Dades i als documents
final FirebaseDatabase db = FirebaseDatabase.getInstance();
final DocumentReference docRef = db.collection("Xat").document("XatProva");
```

observem com no és cap impediment que ens toquem definir-les com a final, ja que aquella referència no ha de canviar en cap moment després d'inicialitzar-la. Si que podem fer canviar el seu valor, però la referència sempre ha d'apuntar al mateix lloc.

KOTLIN

L'equivalent en Kotlin serà:

```
// Referències a la Base de Dades i als documents
val db = FirebaseDatabase.getInstance()
val docRef = db.collection("Xat").document("XatProva")
```

Guardar dades

L'operació de guardar ara consistirà en actualitzar un document. Tindrem 3 maneres d'actualitzar-lo:

- Subscriurem tot el document amb el mètode `set()`
- Editem-lo tot amb el mètode `update()`
- Modifiquem-lo amb el mètode `update()`

Excepte per elaborar, per a les altres operacions ens fa falta saber l'estructura del document. Per això tant `set()` com el mètode `update()` accepten com a paràmetre no una única dada, sinó una estructura que puga ambar a reflectir el document. Acceptaran com a paràmetre un `Map<String, Object>`, on podem col·locar les claus i els valors de tots els membres del document. Cada valor pot ser dels tipus que vam practicar en el punt anterior: string, number, boolean, array, map, ...

La manera de col·locar un element en una estructura `Map<String, Object>` que acceptarà els paràmetres: la clau i el valor.

En el nostre exemple guardarem en el moment d'apretar el botó. Inicialment ho farem sobre `Xat/XatProva/ultimUsuari` i `Xat/XatProva/ultimoMessage`, encara que després ho canviarem

JAVA

```
// Per a guardar dades
// Primer sobre /Xat/XatProva/ultimoUserari / Xat/XatProva/ultimoMessage
// Després tindrem com a documents en la col·lecció /Xat/XatProva/messages
boton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        String text = textInput.getText().toString();
        Map<String, Object> dades = new HashMap<String, Object>();
        dades.put("ultimoUserari", user.getUid());
        dades.put("ultimoMessage", text);
        dades.put("ultimoMessage", text.getText().toString());
        docRef.update(dades);
        text.setText("");
    }
});
```

KOTLIN

```
// Per a guardar dades
// Primer sobre /Xat/XatProva/ultimoUserari / Xat/XatProva/ultimoMessage
// Després tindrem com a documents en la col·lecció /Xat/XatProva/messages
boton.setOnClickListener {
    val dades = HashMap<String, Any>
    dades["ultimoUserari"] = user.text.toString()
    dades["ultimoMessage"] = text.text.toString()
    dades["ultimoMessage"] = text.text.toString()
    docRef.update(dades)
    text.setText("")
}
```

Recuperar dades

També ens plantejarem els dos casos que ja ens vam plantejar en Realtime Database:

- Una única lectura
- Un listener que es quede escoltant per si hi ha canvis

Listener de lectura única: addOnSuccessListener()

Ara des d'Android, per fer la lectura únicament ens plantejarem un listener, que quedarà més sòlid i que podrem actuar quan estiguem segurs que la dada ha arribat.

En realitat tenim més d'un listener per a comprender com ha anat una lectura:

- `addOnSuccessListener()` activarà quan la tasca de lectura finalitza i forma part de la seva execució.
- `addOnCompleteListener()` activarà quan la tasca de lectura finalitza i forma part de la seva execució.
- `addOnSuccessListener()` activarà quan la tasa de lectura ha finalitzat, de forma exhaustiva o no. El més normal serà comprovar dins d'ell si ha anat bé.

Per simplificar, en aquest moment només tractarem el primer d'ells, ja que ens assegura que la lectura ha anat bé. Quan això hoaja passar (`onSuccess()`) podem utilitzar:

- `getRead()` per a obtenir tot el document en un `Map<String, Object>`
- `get()` per a obtenir el nom del document
- `getUid()` per a obtenir directament el valor d'un dels documents
- `getDocumentSnapshot()` per a obtenir el valor de la classe `DocumentSnapshot`
- `getDocumentSnapshot()` per a obtenir el valor de la dada en forma de dòuble
- `getBoolean()`

L'exemple d'única lectura i farà per a posar el títol de l'aplicació, com en l'exemple del Realtime Database.

I aquestes seran les sentències per a posar el títol de l'aplicació:

JAVA

```
// Exemple de lectura única: AddOnSuccessListener()
// Per a posar el títol. Sobre /Xat/XatProva/nomCat
docRef.get().addOnSuccessListener(new OnSuccessListener<DocumentSnapshot>() {
    @Override
    public void onSuccess(DocumentSnapshot documentSnapshot) {
        String nomCat = documentSnapshot.getString("nomCat");
        setTitle(nomCat);
    }
});
```

KOTLIN

```
// Exemple de lectura única: AddOnSuccessListener()
// Per a posar el títol. Sobre /Xat/XatProva/nomCat
docRef.get().addOnSuccessListener { documentSnapshot ->
    setTitle(documentSnapshot.getString("nomCat"))
}
```

Listener que es queda escoltant: addSnapshotListener()

De forma similar al Realtime Database, si volem rebre una notificació de quan hoaja un canvi en el document que ens interessa, sobre una referència a aquest document ens muntarem un listener, en aquest cas amb el mètode `addSnapshotListener()`. Al mètode `onEvent()` que s'ha de sobrescriure arribarà un paràmetre del tipus `DocumentSnapshot`, que serà una còpia del document. Com en el cas anterior podríem fer sobre ell un `getData()` per a obtenir el document.

En el nostre exemple, de moment l'utilitzem tant per a posar l'últim missatge com per a anar omplint l'àrea central amb el sat.

JAVA

```
// Exemple d'listener de lectura continua addSnapshotListener() sobre un document
// Per a posar l'últim missatge registrat. Sobre /Xat/XatProva/ultimoUserari / Xat/XatProva/ultimoMessage
docRef.addSnapshotListener(new EventListener<DocumentSnapshot>() {
    @Override
    public void onEvent(@Nullable DocumentSnapshot documentSnapshot, @Nullable FirebaseFirestoreException e) {
        ultimeText = documentSnapshot.getString("ultimoUserari");
        ultimeText = documentSnapshot.getString("ultimoMessage");
        area.append(documentSnapshot.getString("ultimoUserari") + ":" + documentSnapshot.getString("ultimoMessage") + "\n");
    }
});
```

El més desitjable seria trigar els errors, cosa que no hem fet en aquest moment per simplicitat.

KOTLIN

```
// Exemple de listener de lectura continua addSnapshotListener() sobre un document
// Per a posar l'últim missatge registrat. Sobre /Xat/XatProva/ultimoUserari / Xat/XatProva/ultimoMessage
docRef.addSnapshotListener { documentSnapshot, exception ->
    ultimeText = documentSnapshot.getString("ultimoUserari");
    ultimeText = documentSnapshot.getString("ultimoMessage");
    area.append(documentSnapshot.getString("ultimoUserari") + ":" + documentSnapshot.getString("ultimoMessage") + "\n");
}
```

Guardar documents

Ja sabem que l'estructura de col·leccions i documents, i que dins d'un document pot guardar col·leccions, en les quals hi haurà documents, ... amb una estructura que pot ser recursiva, convéria organitzar d'aquesta manera la informació, en compte d'utilitzar llistes (que també es pot però no són tan pràctiques), millor organitzar-ho en forma de subcol·leccions i documents. Així anem a repetir el ja fet en el moment de Java, adaptant-lo a Android.

Per tant ens és necessària l'operació d'afegir un document a una col·lecció. Això en un principi ho aconseguirem amb el mètode `set()` sobre un document nou de la col·lecció:

```
database.collection("nomCol").document("nomDoc").set(dades);
```

però això obligaria a posar un nom a cada document. Ja havíem vist que Cloud Firestore era capaç de generar un nom de document que no es puga repetir. Dels de Java s'aconsegueix això amb el mètode `add()`:

```
database.collection("nomCol").add(dades);
```

però que en el nostre cas, per ser una subcol·lecció és un poc més llarg:

```
database.collection("Xat").document("XatProva").collection("missatges").add(dades);
```

L'estructura de les dades la podem fer amb un `Map<String, Object>`, i posar-hi en el nostre cas l'usuari i el missatge. D'aquesta manera ens quedaria així el procediment en apretar el botó d'enviar el missatge, on a banda del que teníem abans per a modificar `ultimoUserari` i `ultimoMessage`, ens alegrem el document nou.

JAVA

```
// Per a guardar dades
// Primer sobre /Xats/XatProv/ultimmissatge | /Xats/XatProv/ultimmissatge
// Darrere es posa com a document en la col·lecció /Xats/XatProv/missatges
bot.setOnChildListener(new ValueEventListener() {
    @Override
    public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
        Map<String, Object> dades = new HashMap<>();
        dades.put("ultimmissatge", dataSnapshot.getText());
        dades.put("ultimoMessage", dataSnapshot.getText());
        dataSnapshot.getRef().update(dades);
    }
});

Message m = new Message(usuario1.getText(), text.getText());
db.collection("Xats").document(comboXats.getSelectedItem().toString()).collection("missatges").add(m);
text.setText("");
}}
```

KOTLIN

```
// Per a guardar dades
// Primer sobre /Xats/XatProv/ultimmissatge | /Xats/XatProv/ultimmissatge
// Darrere es posa com a document en la col·lecció /Xats/XatProv/missatges
bot.setOnChildListener {
    val dades = HashMap<String, Any>
    dades["ultimoMessage"] = textView.text.toString()
    dades["ultimoMessage"] = textView.text.toString()
    dscRef.update(dades)
}

val m = Message(usuario1.text.toString(), textView.text.toString())
db.collection("Xats").document(comboXats.selectedItem.toString()).collection("missatges").add(m)
text.setText("")}
```

Recuperar documents modificats

Ja només ens queda detectar els canvis en els documents de la col·lecció, per a afegeir a l'assa central els documents alegits. També és un `addSnapshotListener()`, però ara l'apliquem a una col·lecció (no a un document). El resultat es que d'aquesta forma podem detectar els documents alegits, els modificats i tots els elaborats.

Així tenim el fragment de programa que ens ho permetrà. En aquest exemple només hem detectat el cas de document alegit, i no hem posat els casos de document modificat i document elaborat, ja que aquest exemple no ho utilitzarà.

JAVA

```
// Exemple de llistar de lectura continua addSnapshotListener() sobre una col·lecció
// Per a posar tota la llista de missatges. Sobre /Xats/XatProv/missatges
db.collection("Xats").document("XatProv").collection("missatges").addSnapshotListener(new EventListener<QuerySnapshot>() {
    @Override
    public void onEvent(@Nullable QuerySnapshot snapshots, @Nullable FirebaseFirestoreException e) {
        for (DocumentChange dc : snapshots.getDocumentChanges()) {
            switch(dc.getType()) {
                case ADDED:
                    area.append(dc.getDocument().getString("nom") + ":" + dc.getDocument().getString("contingut") + "\n");
                    break;
            }
        }
    }
});
```

KOTLIN

```
// Exemple de llistar de lectura continua addSnapshotListener() sobre una col·lecció
// Per a posar tota la llista de missatges. Sobre /Xats/XatProv/missatges
db.collection("Xats").document("XatProv").collection("missatges").addSnapshotListener { snapshots, e ->
    for (DocumentChange dc : snapshots.getDocumentChanges()) {
        when (dc.type) {
            DocumentChange.Type.ADDED -> {
                area.append(dc.document.getString("nom") + ":" + dc.document.getString("contingut") + "\n")
            }
        }
    }
}
```

Es de destacar que ara la còpia de les dades, el snapshot, és en realitat un `QuerySnapshot`. I és que nosaltres no hem fet cap consulta sobre els documents de la col·lecció, però és possible fer-la. Per exemple es podria seleccionar per mig d'una query tots els documents en què l'usuari és un determinat. Aleshores detectaríem els documents alegits, modificats o elaborats d'aquest usuari. Ho faríem en el moment de declarar el `addSnapshotListener()`:

```
db.collection("Xats").document("XatProv").collection("missatges").whereEqualTo("usuari", "Usuari1").addSnapshotListener(new EventListener<QuerySnapshot>() {
```

O també podríem ordenar els documents per algun camp. Aqüí en realitat seria necessari per a poder ordenar els missatges, ja que el nom del document auto-generat per Cloud Firestore no té successor, sinó aleatori, i per tant perfectament un document nou tinga un nom anterior a alguns dels existents. Ho arreglarem fàcilment posant un camp més amb la data i ordenant per aquest camp amb `orderBy()` (que anirà en el lloc del `whereEqualto()` de la secció anterior). Ho farem en l'exemple complet de la següent pàgina.

Un altre exemple. Anem a fer una lectura ònica de tots els documents de la col·lecció principal `Xats`. Fins al moment hem treballat sempre sobre el document `XatProv`, però hi anem a generalitzar per a accedir a més d'un xat.

De moment ens aconseguirem amb fer una lectura ònica de tots els documents del xat per a mostrar-los en un Spinner. Posteriorment, en l'exemple complet de la següent pàgina intentarem tirar un xat o un altre per a que ens mostri els missatges de l'un o de l'altre.

Per a fer aquesta lectura ònica, ara utilitzarem un `addOnCompleteListener` en compte del `addOnSuccessListener` de l'altra ocasió, i així veurem un poc la diferència de plantejament.

JAVA

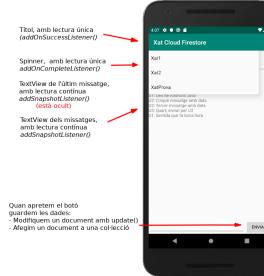
```
// Exemple de llegir tots els documents d'una col·lecció
// Per a trinxar el xat
db.collection("Xats").get().addOnCompleteListener(new OnCompleteListener<QuerySnapshot>() {
    @Override
    public void onComplete(@NonNull Task<QuerySnapshot> task) {
        public void onComplete(@NonNull Task<QuerySnapshot> task) {
            if (task.isSuccessful()) {
                for (QueryDocumentSnapshot document : task.getResult()) {
                    options.add(document.getId());
                }
            } else {
                ArrayAdapter<String> adapter = new ArrayAdapter<(String)>(parentPrincipal, android.R.layout.simple_spinner_item, options);
                adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
                comboXats.setAdapter(adapter);
            }
        }
    }
});
```

Observem que des del mètode `onComplete()` (en compte del `onSuccess()`) mirem ara si s'ha completat satisfactoriàmient la tasca amb `task.isSuccessful()`. Per simplicitat ara no fem res si no ha tingut llectura. Únicament tractem quan si que ha tingut exit, que aprofitem per a omplir el Spinner a partir d'un `ArrayAdapter`.

KOTLIN

```
// Exemple de llegir tots els documents d'una col·lecció
// Per a trinxar el xat
db.collection("Xats").get().addOnCompleteListener { task ->
    if (task.isSuccessful()) {
        val documents = task.getResult()
        for (DocumentSnapshot document in task.getResult()) {
            options.add(document.id)
        }
    } else {
        val adapter = ArrayAdapter<(String)>(parentPrincipal, android.R.layout.simple_spinner_item, options)
        adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item)
        comboXats.setAdapter(adapter)
    }
}
```

En aquesta imatge es veu el resultat, on s'intenta explicar cada cosa de les coses:



De moment el Spinner no fa res quan seleccionem una opció.

Hi intentarem en l'exemple complet de la següent pàgina.

2.3.3.3 CF-Android: Tot l'exemple

- Tindrem la classe **Misatge** amb les propietats **nom** (contingut) (expressió d'una de les noms utilitzats en l'exemple anterior per poder comprovar que aquestes modificacions funcionen i mitjónen l'exemple). Posarem també la **data** (com un long) per a poder ordenar els missatges cronològicament

- Llevem les coses que no s'utilitzaran estípticament en aquest exemple
- El Spiner anomenat **comboKata** ja an funciona. Per a no modificar les coses ja fetes, posarem la creació dels listeners en un mètode anomenat **initializar** després trirem un altre xarà i caldrà els listeners una altra vegada per a que apunten a lloc correcte, els tindrem per duplicat. Hauríem de parlar els primers listeners.

1

JAVA

```
Aguedo class Message

public class Message{
    public String nom;
    public long data;
    public String contingut;

    public Message(){
    }

    public Message(String nom, String cont) {
        this.nom = nom;
        this.contingut = cont;
    }

    public Message(String nom, long data, String contingut) {
        super();
        this.nom = nom;
        this.data = data;
        this.contingut = contingut;
    }

    public String getNom() {
        return nom;
    }

    public void setNom(String nom) {
        this.nom = nom;
    }

    public long getData() {
        return data;
    }

    public void setData(long data) {
        this.data = data;
    }

    public String getContingut() {
        return contingut;
    }

    public void setContingut(String contingut) {
        this.contingut = contingut;
    }
}
```

```
// Per a guardar dades
// Primer sobre /Xats/XatProv/ultimMissatge i /Xats/XatProv/ultimMissatge
// Després també com a documents en la col·lecció /Xats/XatProv/missatges
dbato.setOnChildListener(new View.OnChildListener() {
    @Override
    public void onChildClick(View view) {
        Map<String, Object> dades = new HashMap<>();
        dades.put("ultimoUser", ultimoUser);
        dades.put("ultimoTit", ultimoTit);
        dades.put("ultimoContingut", text.getText());
        docRef.update(dades);

        Missatge m = new Missatge(usuario.getText().toString(), new Date().getTime(), text.getText().toString());
        db.collection("Xats").document(comboXats.getSelectedItem().toString()).collection("missatges").add(m);
    }
});

@Override
public void onNothingSelected(AdapterView<?> arg0) {
    // TODO Auto-generated method stub
}
});
}
}
}
```

KOTLIN

Classe Missatge

```
class Missatge(val nom: String, val data: Long, val contingut: String) {}

El programa:

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.view.View
import android.widget.*
import com.google.firebase.firestore.*
import java.util.SimpleDateFormat
import java.util.*

import kotlin.android.synthetic.main.activity_main.*

class MainActivity : AppCompatActivity() {

    var listenerUltimMissatge: ListenerRegistration? = null
    var listenerMissatges: ListenerRegistration? = null

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        botó.text = "Enviar"
        val pantPrincipal = this

        // Referències a la Base de Dades als documents
        val db = FirebaseFirestore.getInstance()
        var docRef: DocumentReference = db.collection("Xats").document("XatProv");

        // Exemple de llegir tots els documents d'una col·lecció
        // Per a triar el xat
        db.collection("Xats").addOnCompleteListener { task ->
            if (task.isSuccessful) {
                val options = ArrayList<String>
                for (doc in task.result!!) {
                    options.add(doc.id)
                }
                val adaptador =
                    ArrayAdapter(pantPrincipal, android.R.layout.simple_spinner_item, options)
                adaptador.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item)
                comboXats.adapter = adaptador
            } else {
            }
        }

        comboXats.onItemSelectedListener = object AdapterView.OnItemSelectedListener {
            override fun onItemSelected(arg0: AdapterView<?>, arg1: View, arg2: Int, arg3: Long) {
                // TODO Auto-generated method stub
                var db = db.collection("Xats").document(comboXats.selectedItem.toString())
                area.text = ""
                inicialitzar()
            }
        }

        private fun inicialitzar() {
            // Exemple de lectura inicial: addOnSuccessListener()
            // Per a posar el títol. Sobre /Xats/XatProv/ultimoXat
            docRef.get().addOnSuccessListener { documentSnapshot: DocumentSnapshot? ->
                if (documentSnapshot != null) {
                    title = documentSnapshot.getString("ultimoXat")
                }
            }

            // Exemple de llistar de lectura continua addOnSuccessListener() sobre un document
            // Per a posar l'últim missatge registrat. Sobre /Xats/XatProv/ultimoMissatge
            // Si es vol fer una lectura continua, cal mantenir el cursor per a tornar-lo a llançar
            if (listenerUltimMissatge == null) {
                listenerUltimMissatge = docRef.addSnapshotListener { documentSnapshot, e ->
                    ultimoTit = documentSnapshot.getString("ultimoTit")
                    ultimoContingut = documentSnapshot.getString("ultimoContingut")
                    ultimoUser = documentSnapshot.getString("ultimoUser")
                }
            }

            // Exemple de llistar de lectura continua addOnSuccessListener() sobre una col·lecció
            // Per a posar tota la llista de missatges. Sobre /Xats/XatProv/missatges
            if (listenerMissatges == null) {
                listenerMissatges = db.collection("Xats").document(comboXats.selectedItem.toString()).collection("missatges").addSnapshotListener { snapshots, e ->
                    for (dc in snapshots!!) {
                        when (dc.documentChangeType) {
                            DocumentChangeType.ADDED -> {
                                val sdf = SimpleDateFormat("dd-MM-yyyy HH:mm")
                                val dataFormatada = sdf.format(dc.document.getString("data")!!)
                                area.append("\n" + dc.document.getString("ultimoUser") + " " + dataFormatada + " " + dc.document.getString("contingut") + "\n")
                            }
                        }
                    }
                }
            }

            // Per a guardar dades
            // Primer sobre /Xats/XatProv/ultimMissatge i /Xats/XatProv/ultimMissatge
            // Després també com a documents en la col·lecció /Xats/XatProv/missatges
            botó.setOnClickListener {
                val dades = HashMap<String, Any>()
                dades["ultimoUser"] = ultimoUser
                dades["ultimoTit"] = ultimoTit
                dades["ultimoContingut"] = text.text.toString()
                docRef.update(dades)

                val m = Missatge(usuario.getText().toString(), Date().time, text.getText())
                db.collection("Xats").document(comboXats.getSelectedItem().toString()).collection("missatges").add(m)

                text.setText("")
            }
        }

        override fun onNothingSelected(arg0: AdapterView<?>) {
            // TODO Auto-generated method stub
        }
    }
}
}
```

I el resultat tant en un cas com en l'altre és el de la imatge, en la qual es veu el contingut del xat Xat1.



Segurament de tots els tipus de Bases de Dades NoSQL existents, la de més fàcil comprendre és la **Base de Dades Clau-Valor**. En ella s'anicten guardant parelles clau-valor, és a dir, el nom d'una determinada propietat i el seu valor. La clau ha de ser única, és a dir, no es pot repetir ja que en cas contrari no es podrà tornar a recuperar.

Per tant no hi ha definició de taules ni cap altra estructura, es ven guardant parelles clau-valor i prou. En el moment de recuperar la informació veurem que podem obtenir el contingut d'una clau, o el de més d'una al mateix temps.

L'exemple que veurem de Base de Dades Clau-Valor és **Redis**, molt famosa per a seua potència i eficiència.

La clau sempre és de tipus String, i com ja hem comentat no podem haver dues claus iguals. En Redis els valors poden ser de 5 tipus diferents:

- **Cadenes de caràcters (String)**: Per exemple: nom, 1 → "Albert"
- **Enteros (Integer)**: Per exemple: quantitat d'elements dins d'una llista-clau. Per exemple: empleat_1 → 1 nom="Albert", departament="10", sour="1000.0"
- **Listes (List)** o conjunts d'elements ordenats de valors. Per exemple: lista_1 → ["Primer", "Segon", "Tercer"]
- **Conjunts (Sets)** són conjunts desordenats de valors. No importa el seu ordre, i de fet seria impossible saber amb el qual els torna Redis. Per exemple: colors → {"Blau", "Verd", "Roig"}
- **Conjunts Ordenats (Sorted Sets)**, que intenten reunir els avantatges dels conjunts, però que seran ordenats. Ja veurem la diferència entre llistes i conjunts ordenats.

Algunes característiques de Redis són:

- És una arquitectura client-server.
- Els clients no estan obligats a fer canviar tota la memòria, encara que si no pot canviar-la també funcionarà de forma molt ràpida. I a més manté una sincronització constant a disc per a fer les dades persistentes. Aquesta tasa la fa en segon pis, de manera que no afecta al servei.
- Per a poder suportar ratios de lectura molt alta i una replicació master/slave, és a dir que per haver més d'un servidor, i un és el que actual de master i els altres són réplicues del primer. El slave rep una copia inicial de la Base de Dades senzilla. A més que es van realitzant escriptures en el master, es van enviant a tots els slaves connectats. Els clients es poden connectar als distints servidors, bé siga al master o als slaves, sense haver de canviar sempre al master.

3.1 - Instal·lació de Redis

Redis està construït per a Linux. També funciona, però, des de Windows com veurem una miqueta més aviat.

Instal·lació en Linux

El lloc des d'on baixar és la pàgina oficial:

<https://redis.io/>

En el moment de fer aquestes captures, l'última versió estable és la 4.0.6.

Era solament el fitxer .tar.gz i descomprimiu, i després d'un terminal ens situem en el directori on ha descomprimit i fem make per a generar els executables. Després de molts avisos, s'autrifica l'error instal·lat. Aquest seria el resultat d'acions, fets tots ells des d'una terminal (però no cal descomprimir des d'una terminal) i havent-nos situat prèviament en el lloc on està el fitxer baixat. També suposem que el fitxer està col·locat en el lloc on volem que estigui instal·lat de forma definitiva. Recordeu que podreu descomprimir-lo de la manera que us resulta més cómoda.

```
tar xf redis-4.0.6.tar.gz
cd redis-4.0.6
make
```

Amb això s'haurien d'haver generat els executables. I ja hauria de funcionar.

Per a posar en marxa el servidor, quasi que el més cómoda serà obrir un terminal, situar-nos en el directori **redis-4.0.6/src** i des d'allí executar **redis-server**. Haurà d'haver una finestra similar a la següent, amb més o menys avisos (observeu que al principi de la imatge estan les entrades donades; i en la imatge està per a la versió 3.0.7, però per a la versió actual es completement equivalent):

```
*:~$ redis-3.0.7/redis-server
127.0.0.1:6379] 17 Feb 12:46:09.076 # WARNING: The TCP backlog setting of 511 cannot be enforced because /proc/sys/net/ipv4/tcp_max_syn_backlog is set to the lower value of 128.
127.0.0.1:6379] 17 Feb 12:46:09.076 # Server started, Redis version 3.0.7
127.0.0.1:6379] 17 Feb 12:46:09.076 # Configuration loaded from '/etc/redis/redis.conf'
127.0.0.1:6379] 17 Feb 12:46:09.076 # Redis is running on port 6379.
127.0.0.1:6379] 17 Feb 12:46:09.076 # You requested maxclients of 10000, requiring at least 81932 Max file descriptors.
127.0.0.1:6379] 17 Feb 12:46:09.076 # Redis can't set maximum open files to 10000 because of OS error: Operation not permitted.
127.0.0.1:6379] 17 Feb 12:46:09.076 # Redis is now running with maxclients of 4096, maxclients has been reduced to 4096 to compensate for low ulimit. If you need higher maxclients increase 'ulimit -n'.
127.0.0.1:6379] 17 Feb 12:46:09.076 # Redis 3.0.7 (00000000/0) 64 bit
127.0.0.1:6379] 17 Feb 12:46:09.076 # Running in standalone mode
127.0.0.1:6379] 17 Feb 12:46:09.076 # PID: 7619
127.0.0.1:6379] 17 Feb 12:46:09.076 # http://redis.io

Redis 3.0.7 (00000000/0) 64 bit
Running in standalone mode
Port: 6379
PID: 7619
http://redis.io
```

Entre altres coses dóna que el servidor està en marxa esperant connexions al port 6379, que és el port per defecte de Redis. Aquesta finestra del terminal l'hauríem de deixar en marxa. Quan vulguem detenir Redis, senzillament fem **ctrl-c**, i deixarem l'excepció de forma ordenada (guardant-se les dades no guardades).

Podrihem executar directament **redis-server** fent doble clic des d'un explorador d'àrboles, per exemple, però aleshores no podríem parar-lo i en definitiva controlar-lo tan cómodament.

Per a fer una connexió des d'un client, també des d'un terminal (un altre) executem **redis-cli**:

```
*:~$ redis-3.0.7/redis-cli
127.0.0.1:6379>
```

Ja ha fet la connexió, concretament a localhost (127.0.0.1) al port 6379, que havíem quedat que és el port per defecte.

Comproveu que si funciona. Encara no hi ha dades, perquè l'acabem d'instalar. I recordeu que és una Base de Dades clau-valor. Per crear una entrada posarem **set clau valor**. Per a obtenir la posarem **get clau**. En la imatge es pot comprovar:

```
*:~$ redis-3.0.7/redis-cli
127.0.0.1:6379> set clau_1
127.0.0.1:6379> get clau_1
127.0.0.1:6379>
```

Hem creat una clau anomenada **clau_1** amb el valor primera, com es pot comprovar en el moment d'obtenir-la amb **get**.

Si al programar **redis-cli** no s'posen paràmetres, intentarà fer una connexió local (localhost). Si volem connectar a un servidor situat en una altra adreça, li la possem amb el paràmetre **-h adreça**, per exemple:

```
redis-3.0.7/redis> -h 192.168.1.24
```

Ac tenim una imatge des d'una connexió externa, des d'un altre equip:

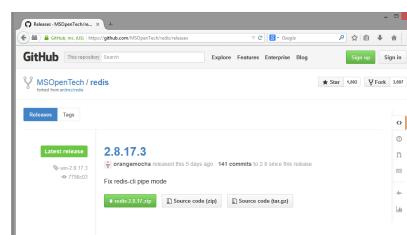
```
alvaro@alvaro: ~redis-2.8.17/redis>
alvaro@alvaro: ~redis-2.8.17/redis> ./redis-cli -h 192.168.1.24
192.168.1.24:6379> get clau_1
192.168.1.24:6379> 192.168.1.24:6379>
```

Instal·lació en Windows de 64 bits

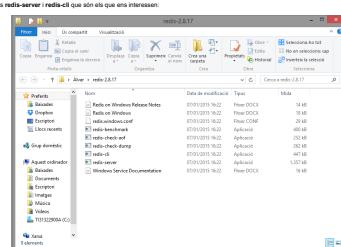
Encara que Redis està construït per a Linux, hi ha versions per Windows, preferiblement de 64 bits. També podem trobar versions de 32 bits, però molt més antigues.

El lloc on poder baixar els fitxers de Redis per a Windows de 64 bits és:

<https://github.com/MSOpenTech/redis/releases>



Era baixem el zip, el descomprimiu, i ja ho tindrem disponible (sense fer **make** ni res). Observeu com en la carpeta resultat de descomprimir ja tenim els executables **redis-server** i **redis-cli** que són els que ens interessen:



Executem **redis-server** directament i ja el tindrem en marxa:



Executem també el [redis-cli](#) i el resultat serà el mateix que en Linux.

[Instal·lació en Windows de 32 bits](#)

És un poc més complicada de trobar. I sobretot, és una versió prou més antiga. Podem descarregar-la de la següent adreça:

<https://github.com/iglesiaslfernando>

Aquesta si que és d'instal·lació (no de descompressió únicament). La documentació suggerixa que s'active com un servei. Per això, en la llista de programes de l'inici no està el servidor, únicament el client. Però el podem trobar en el directori on s'ha instal·lat: C:\Archives de programa\Redis. Trobarem tant redis-server.exe com redis-cli.exe.

Anem a veure la utilització de Redis. Ens connectarem com a clients i intentarem fer operacions.

- Les primeres seran les més senzilles, utilitzant únicament el tipus de dades String.
- Posteriorment mirarem com treballar amb les clau: buscar-ne una, veure si existeix, buscar unes quantes, ...
- Després ja anirem a pel·lis tipus de dades més complicats:
 - List
 - Set
 - Sorted Set

Decrementa en una unitat el valor de la clau (sempre que siga un enter).

Per agafar valors negatius.

Si la clau no existeix la crea assumint que valia 0, i per tant després valdrà -1.

```
[127.0.0.1:51235] > decr compt2  
[127.0.0.1:51235] >  
[127.0.0.1:51235] > decr compt2  
[127.0.0.1:51235] > get compt2  
[127.0.0.1:51235] >
```

INCRBY

Status

```
[incrby clau1 increment]
```

Incrementa el valor de la clau en el numero d'unitats indicat en increment (el valor ha de ser enter). L'increment pot ser negatiu.

```
[127.0.0.1:51235] > incrby compt1 10  
[127.0.0.1:51235] > incrby compt1 -20  
[127.0.0.1:51235] > incrby compt1 5  
[127.0.0.1:51235] >
```

DECRBY

Status

```
[decrby clau1 decrement]
```

Decrementa el valor de la clau el numero d'unitat indicat en decrement.

```
[127.0.0.1:51235] > decrby compt1 5  
[127.0.0.1:51235] >
```

3.2.2 - Keys

Ara això a veure comandos que ens permeten treballar amb les claus, per a buscar-les, veure si existeixen, etc. No importarà el tipus de les claus (de moment només hem treballat amb claus de tipus `String`, però si ja en tinguem d'altres tipus també es veuràn afectades). En cas que d'aquests comandos accedirem al valor de les claus.

KEYS**Sintax****keys patro**

Torna totes les claus que coincideixen amb el patró. En el patró podem posar caràcters comodí:

- *: només un o més caràcters. Per exemple "Mara*" podria tornar "Mara", "Maria", "Margalida", ...
- ?: equivalente a un caràcter. Per exemple "Mar?a" podria tornar "Mara" o "Marta", però no "Mesa", "Margalida".
- [ab]: serà cert si en el lloc corresponent hi ha un dels caràcters especificats entre els parèntesis. Per exemple "Mar[ab]" podria tornar "Maria" o "Marta", però no "Marga".

Per a tornar totes les claus utilitzarem `keys *`:

```
[127.0.0.1:4379> keys *
1) "compt1"
2) "clau_1"
3) "clau_2"
4) "compt3"
5) "clau_3"
6) "mara2"
7) "mara1"
8) "salutacio"
9) "mara3"
10) "mara1"
11) "mara2"
12) "mara3"
13) "compt2"
14) "clau_25"
15) "clau_123"
16) "mara4"
17) "mara5"
18) "mara6"
19) "mara7"
20) "mara8"
21) "mara9"
22) "mara10"
23) "mara11"
24) "mara12"
25) "mara13"
26) "mara14"
27) "mara15"
28) "mara16"
29) "mara17"
30) "mara18"
31) "mara19"
32) "mara20"
33) "mara21"
34) "mara22"
35) "mara23"
36) "mara24"
37) "mara25"
38) "mara26"
39) "mara27"
40) "mara28"
41) "mara29"
42) "mara30"
43) "mara31"
44) "mara32"
45) "mara33"
46) "mara34"
47) "mara35"
48) "mara36"
49) "mara37"
50) "mara38"
51) "mara39"
52) "mara40"
53) "mara41"
54) "mara42"
55) "mara43"
56) "mara44"
57) "mara45"
58) "mara46"
59) "mara47"
60) "mara48"
61) "mara49"
62) "mara50"
63) "mara51"
64) "mara52"
65) "mara53"
66) "mara54"
67) "mara55"
68) "mara56"
69) "mara57"
70) "mara58"
71) "mara59"
72) "mara60"
73) "mara61"
74) "mara62"
75) "mara63"
76) "mara64"
77) "mara65"
78) "mara66"
79) "mara67"
80) "mara68"
81) "mara69"
82) "mara70"
83) "mara71"
84) "mara72"
85) "mara73"
86) "mara74"
87) "mara75"
88) "mara76"
89) "mara77"
90) "mara78"
91) "mara79"
92) "mara80"
93) "mara81"
94) "mara82"
95) "mara83"
96) "mara84"
97) "mara85"
98) "mara86"
99) "mara87"
100) "mara88"
101) "mara89"
102) "mara90"
103) "mara91"
104) "mara92"
105) "mara93"
106) "mara94"
107) "mara95"
108) "mara96"
109) "mara97"
110) "mara98"
111) "mara99"
112) "mara100"
113) "mara101"
114) "mara102"
115) "mara103"
116) "mara104"
117) "mara105"
118) "mara106"
119) "mara107"
120) "mara108"
121) "mara109"
122) "mara110"
123) "mara111"
124) "mara112"
125) "mara113"
126) "mara114"
127) "mara115"
128) "mara116"
129) "mara117"
130) "mara118"
131) "mara119"
132) "mara120"
133) "mara121"
134) "mara122"
135) "mara123"
136) "mara124"
137) "mara125"
138) "mara126"
139) "mara127"
140) "mara128"
141) "mara129"
142) "mara130"
143) "mara131"
144) "mara132"
145) "mara133"
146) "mara134"
147) "mara135"
148) "mara136"
149) "mara137"
150) "mara138"
151) "mara139"
152) "mara140"
153) "mara141"
154) "mara142"
155) "mara143"
156) "mara144"
157) "mara145"
158) "mara146"
159) "mara147"
160) "mara148"
161) "mara149"
162) "mara150"
163) "mara151"
164) "mara152"
165) "mara153"
166) "mara154"
167) "mara155"
168) "mara156"
169) "mara157"
170) "mara158"
171) "mara159"
172) "mara160"
173) "mara161"
174) "mara162"
175) "mara163"
176) "mara164"
177) "mara165"
178) "mara166"
179) "mara167"
180) "mara168"
181) "mara169"
182) "mara170"
183) "mara171"
184) "mara172"
185) "mara173"
186) "mara174"
187) "mara175"
188) "mara176"
189) "mara177"
190) "mara178"
191) "mara179"
192) "mara180"
193) "mara181"
194) "mara182"
195) "mara183"
196) "mara184"
197) "mara185"
198) "mara186"
199) "mara187"
200) "mara188"
201) "mara189"
202) "mara190"
203) "mara191"
204) "mara192"
205) "mara193"
206) "mara194"
207) "mara195"
208) "mara196"
209) "mara197"
210) "mara198"
211) "mara199"
212) "mara200"
213) "mara201"
214) "mara202"
215) "mara203"
216) "mara204"
217) "mara205"
218) "mara206"
219) "mara207"
220) "mara208"
221) "mara209"
222) "mara210"
223) "mara211"
224) "mara212"
225) "mara213"
226) "mara214"
227) "mara215"
228) "mara216"
229) "mara217"
230) "mara218"
231) "mara219"
232) "mara220"
233) "mara221"
234) "mara222"
235) "mara223"
236) "mara224"
237) "mara225"
238) "mara226"
239) "mara227"
240) "mara228"
241) "mara229"
242) "mara230"
243) "mara231"
244) "mara232"
245) "mara233"
246) "mara234"
247) "mara235"
248) "mara236"
249) "mara237"
250) "mara238"
251) "mara239"
252) "mara240"
253) "mara241"
254) "mara242"
255) "mara243"
256) "mara244"
257) "mara245"
258) "mara246"
259) "mara247"
260) "mara248"
261) "mara249"
262) "mara250"
263) "mara251"
264) "mara252"
265) "mara253"
266) "mara254"
267) "mara255"
268) "mara256"
269) "mara257"
270) "mara258"
271) "mara259"
272) "mara260"
273) "mara261"
274) "mara262"
275) "mara263"
276) "mara264"
277) "mara265"
278) "mara266"
279) "mara267"
280) "mara268"
281) "mara269"
282) "mara270"
283) "mara271"
284) "mara272"
285) "mara273"
286) "mara274"
287) "mara275"
288) "mara276"
289) "mara277"
290) "mara278"
291) "mara279"
292) "mara280"
293) "mara281"
294) "mara282"
295) "mara283"
296) "mara284"
297) "mara285"
298) "mara286"
299) "mara287"
300) "mara288"
301) "mara289"
302) "mara290"
303) "mara291"
304) "mara292"
305) "mara293"
306) "mara294"
307) "mara295"
308) "mara296"
309) "mara297"
310) "mara298"
311) "mara299"
312) "mara300"
313) "mara301"
314) "mara302"
315) "mara303"
316) "mara304"
317) "mara305"
318) "mara306"
319) "mara307"
320) "mara308"
321) "mara309"
322) "mara310"
323) "mara311"
324) "mara312"
325) "mara313"
326) "mara314"
327) "mara315"
328) "mara316"
329) "mara317"
330) "mara318"
331) "mara319"
332) "mara320"
333) "mara321"
334) "mara322"
335) "mara323"
336) "mara324"
337) "mara325"
338) "mara326"
339) "mara327"
340) "mara328"
341) "mara329"
342) "mara330"
343) "mara331"
344) "mara332"
345) "mara333"
346) "mara334"
347) "mara335"
348) "mara336"
349) "mara337"
350) "mara338"
351) "mara339"
352) "mara340"
353) "mara341"
354) "mara342"
355) "mara343"
356) "mara344"
357) "mara345"
358) "mara346"
359) "mara347"
360) "mara348"
361) "mara349"
362) "mara350"
363) "mara351"
364) "mara352"
365) "mara353"
366) "mara354"
367) "mara355"
368) "mara356"
369) "mara357"
370) "mara358"
371) "mara359"
372) "mara360"
373) "mara361"
374) "mara362"
375) "mara363"
376) "mara364"
377) "mara365"
378) "mara366"
379) "mara367"
380) "mara368"
381) "mara369"
382) "mara370"
383) "mara371"
384) "mara372"
385) "mara373"
386) "mara374"
387) "mara375"
388) "mara376"
389) "mara377"
390) "mara378"
391) "mara379"
392) "mara380"
393) "mara381"
394) "mara382"
395) "mara383"
396) "mara384"
397) "mara385"
398) "mara386"
399) "mara387"
400) "mara388"
401) "mara389"
402) "mara390"
403) "mara391"
404) "mara392"
405) "mara393"
406) "mara394"
407) "mara395"
408) "mara396"
409) "mara397"
410) "mara398"
411) "mara399"
412) "mara400"
413) "mara401"
414) "mara402"
415) "mara403"
416) "mara404"
417) "mara405"
418) "mara406"
419) "mara407"
420) "mara408"
421) "mara409"
422) "mara410"
423) "mara411"
424) "mara412"
425) "mara413"
426) "mara414"
427) "mara415"
428) "mara416"
429) "mara417"
430) "mara418"
431) "mara419"
432) "mara420"
433) "mara421"
434) "mara422"
435) "mara423"
436) "mara424"
437) "mara425"
438) "mara426"
439) "mara427"
440) "mara428"
441) "mara429"
442) "mara430"
443) "mara431"
444) "mara432"
445) "mara433"
446) "mara434"
447) "mara435"
448) "mara436"
449) "mara437"
450) "mara438"
451) "mara439"
452) "mara440"
453) "mara441"
454) "mara442"
455) "mara443"
456) "mara444"
457) "mara445"
458) "mara446"
459) "mara447"
460) "mara448"
461) "mara449"
462) "mara450"
463) "mara451"
464) "mara452"
465) "mara453"
466) "mara454"
467) "mara455"
468) "mara456"
469) "mara457"
470) "mara458"
471) "mara459"
472) "mara460"
473) "mara461"
474) "mara462"
475) "mara463"
476) "mara464"
477) "mara465"
478) "mara466"
479) "mara467"
480) "mara468"
481) "mara469"
482) "mara470"
483) "mara471"
484) "mara472"
485) "mara473"
486) "mara474"
487) "mara475"
488) "mara476"
489) "mara477"
490) "mara478"
491) "mara479"
492) "mara480"
493) "mara481"
494) "mara482"
495) "mara483"
496) "mara484"
497) "mara485"
498) "mara486"
499) "mara487"
500) "mara488"
501) "mara489"
502) "mara490"
503) "mara491"
504) "mara492"
505) "mara493"
506) "mara494"
507) "mara495"
508) "mara496"
509) "mara497"
510) "mara498"
511) "mara499"
512) "mara500"
513) "mara501"
514) "mara502"
515) "mara503"
516) "mara504"
517) "mara505"
518) "mara506"
519) "mara507"
520) "mara508"
521) "mara509"
522) "mara510"
523) "mara511"
524) "mara512"
525) "mara513"
526) "mara514"
527) "mara515"
528) "mara516"
529) "mara517"
530) "mara518"
531) "mara519"
532) "mara520"
533) "mara521"
534) "mara522"
535) "mara523"
536) "mara524"
537) "mara525"
538) "mara526"
539) "mara527"
540) "mara528"
541) "mara529"
542) "mara530"
543) "mara531"
544) "mara532"
545) "mara533"
546) "mara534"
547) "mara535"
548) "mara536"
549) "mara537"
550) "mara538"
551) "mara539"
552) "mara540"
553) "mara541"
554) "mara542"
555) "mara543"
556) "mara544"
557) "mara545"
558) "mara546"
559) "mara547"
560) "mara548"
561) "mara549"
562) "mara550"
563) "mara551"
564) "mara552"
565) "mara553"
566) "mara554"
567) "mara555"
568) "mara556"
569) "mara557"
570) "mara558"
571) "mara559"
572) "mara560"
573) "mara561"
574) "mara562"
575) "mara563"
576) "mara564"
577) "mara565"
578) "mara566"
579) "mara567"
580) "mara568"
581) "mara569"
582) "mara570"
583) "mara571"
584) "mara572"
585) "mara573"
586) "mara574"
587) "mara575"
588) "mara576"
589) "mara577"
590) "mara578"
591) "mara579"
592) "mara580"
593) "mara581"
594) "mara582"
595) "mara583"
596) "mara584"
597) "mara585"
598) "mara586"
599) "mara587"
500) "mara588"
501) "mara589"
502) "mara590"
503) "mara591"
504) "mara592"
505) "mara593"
506) "mara594"
507) "mara595"
508) "mara596"
509) "mara597"
510) "mara598"
511) "mara599"
512) "mara600"
513) "mara601"
514) "mara602"
515) "mara603"
516) "mara604"
517) "mara605"
518) "mara606"
519) "mara607"
520) "mara608"
521) "mara609"
522) "mara610"
523) "mara611"
524) "mara612"
525) "mara613"
526) "mara614"
527) "mara615"
528) "mara616"
529) "mara617"
530) "mara618"
531) "mara619"
532) "mara620"
533) "mara621"
534) "mara622"
535) "mara623"
536) "mara624"
537) "mara625"
538) "mara626"
539) "mara627"
540) "mara628"
541) "mara629"
542) "mara630"
543) "mara631"
544) "mara632"
545) "mara633"
546) "mara634"
547) "mara635"
548) "mara636"
549) "mara637"
550) "mara638"
551) "mara639"
552) "mara640"
553) "mara641"
554) "mara642"
555) "mara643"
556) "mara644"
557) "mara645"
558) "mara646"
559) "mara647"
560) "mara648"
561) "mara649"
562) "mara650"
563) "mara651"
564) "mara652"
565) "mara653"
566) "mara654"
567) "mara655"
568) "mara656"
569) "mara657"
570) "mara658"
571) "mara659"
572) "mara660"
573) "mara661"
574) "mara662"
575) "mara663"
576) "mara664"
577) "mara665"
578) "mara666"
579) "mara667"
580) "mara668"
581) "mara669"
582) "mara670"
583) "mara671"
584) "mara672"
585) "mara673"
586) "mara674"
587) "mara675"
588) "mara676"
589) "mara677"
590) "mara678"
591) "mara679"
592) "mara680"
593) "mara681"
594) "mara682"
595) "mara683"
596) "mara684"
597) "mara685"
598) "mara686"
599) "mara687"
500) "mara688"
501) "mara689"
502) "mara690"
503) "mara691"
504) "mara692"
505) "mara693"
506) "mara694"
507) "mara695"
508) "mara696"
509) "mara697"
510) "mara698"
511) "mara699"
512) "mara700"
513) "mara701"
514) "mara702"
515) "mara703"
516) "mara704"
517) "mara705"
518) "mara706"
519) "mara707"
520) "mara708"
521) "mara709"
522) "mara710"
523) "mara711"
524) "mara712"
525) "mara713"
526) "mara714"
527) "mara715"
528) "mara716"
529) "mara717"
530) "mara718"
531) "mara719"
532) "mara720"
533) "mara721"
534) "mara722"
535) "mara723"
536) "mara724"
537) "mara725"
538) "mara726"
539) "mara727"
540) "mara728"
541) "mara729"
542) "mara730"
543) "mara731"
544) "mara732"
545) "mara733"
546) "mara734"
547) "mara735"
548) "mara736"
549) "mara737"
550) "mara738"
551) "mara739"
552) "mara740"
553) "mara741"
554) "mara742"
555) "mara743"
556) "mara744"
557) "mara745"
558) "mara746"
559) "mara747"
560) "mara748"
561) "mara749"
562) "mara750"
563) "mara751"
564) "mara752"
565) "mara753"
566) "mara754"
567) "mara755"
568) "mara756"
569) "mara757"
570) "mara758"
571) "mara759"
572) "mara760"
573) "mara761"
574) "mara762"
575) "mara763"
576) "mara764"
577) "mara765"
578) "mara766"
579) "mara767"
580) "mara768"
581) "mara769"
582) "mara770"
583) "mara771"
584) "mara772"
585) "mara773"
586) "mara774"
587) "mara775"
588) "mara776"
589) "mara777"
590) "mara778"
591) "mara779"
592) "mara780"
593) "mara781"
594) "mara782"
595) "mara783"
596) "mara784"
597) "mara785"
598) "mara786"
599) "mara787"
500) "mara788"
501) "mara789"
502) "mara790"
503) "mara791"
504) "mara792"
505) "mara793"
506) "mara794"
507) "mara795"
508) "mara796"
509) "mara797"
510) "mara798"
511) "mara799"
512) "mara800"
513) "mara801"
514) "mara802"
515) "mara803"
516) "mara804"
517) "mara805"
518) "mara806"
519) "mara807"
520) "mara808"
521) "mara809"
522) "mara810"
523) "mara811"
524) "mara812"
525) "mara813"
526) "mara814"
527) "mara815"
528) "mara816"
529) "mara817"
530) "mara818"
531) "mara819"
532) "mara820"
533) "mara821"
534) "mara822"
535) "mara823"
536) "mara824"
537) "mara825"
538) "mara826"
539) "mara827"
540) "mara828"
541) "mara829"
542) "mara830"
543) "mara831"
544) "mara832"
545) "mara833"
546) "mara834"
547) "mara835"
548) "mara836"
549) "mara837"
550) "mara838"
551) "mara839"
552) "mara840"
553) "mara841"
554) "mara842"
555) "mara843"
556) "mara844"
557) "mara845"
558) "mara846"
559) "mara847"
560) "mara848"
561) "mara849"
562) "mara850"
563) "mara851"
564) "mara852"
565) "mara853"
566) "mara854"
567) "mara855"
568) "mara856"
569) "mara857"
570) "mara858"
571) "mara859"
572) "mara860"
573) "mara861"
574) "mara862"
575) "mara863"
576) "mara864"
577) "mara865"
578) "mara866"
579) "mara867"
580) "mara868"
581) "mara869"
582) "mara870"
583) "mara871"
584) "mara872"
585) "mara873"
586) "mara874"
587) "mara875"
588) "mara876"
589) "mara877"
590) "mara878"
591) "mara879"
592) "mara880"
593) "mara881"
594) "mara882"
595) "mara883"
596) "mara884"
597) "mara885"
598) "mara886"
599) "mara887"
500) "mara888"
501) "mara889"
502) "mara890"
503) "mara891"
504) "mara892"
505) "mara893"
506) "mara894"
507) "mara895"
508) "mara896"
509) "mara897"
510) "mara898"
511) "mara899"
512) "mara900"
513) "mara901"
514) "mara902"
515) "mara903"
516) "mara904"
517) "mara905"
518) "mara906"
519) "mara907"
520) "mara908"
521) "mara909"
522) "mara910"
523) "mara911"
524) "mara912"
525) "mara913"
526) "mara914"
527) "mara915"
528) "mara916"
529) "mara917"
530) "mara918"
531) "mara919"
532) "mara920"
533) "mara921"
534) "mara922"
535) "mara923"
536) "mara924"
537) "mara925"
538) "mara926"
539) "mara927"
540) "mara928"
541) "mara929"
542) "mara930"
543) "mara931"
544) "mara932"
545) "mara933"
546) "mara934"
547) "mara935"
548) "mara936"
549) "mara937"
550) "mara938"
551) "mara939"
552) "mara940"
553) "mara941"
554) "mara942"
555) "mara943"
556) "mara944"
557) "mara945"
558) "mara946"
559) "mara947"
560) "mara948"
561) "mara949"
562) "mara950"
563) "mara951"
564) "mara952"
565) "mara953"
566) "mara954"
567) "mara955"
568) "mara956"
569) "mara957"
570) "mara958"
571) "mara959"
572) "mara960"
573) "mara9
```

3.2.3 - Hash

Ja havíem comentat que el tipus Hash és una espècie de registre, amb subamps (en realitat hauríem de dir sub-clau). Pot tenir qualsevol número de subamps que són de tipus String.

Redira molt efficient en quant a l'espai que ocupen els Hash i sobretot en el temps de recuperació de les dades.

Els comandes que van veure per al String no es poden aplicar al Hash. Tantmateix els comandes del Hash són molt similars a aquells, començant sempre per H.

HSET

Sintax

```
hset clave campo valor
```

Assigna al camp especificat de la clau especificada el valor especificat. Si el valor consta de més d'una paraula, haurà d'anar entre cometes dobles.

Si la clau no existeix, la creaix. I si ja existeix, senzillament afegirà el camp. I si d'aquesta clau ja existia el camp, modificarà el seu valor.

Entendrem, en claus diferents poden haver camps amb els mateixos noms.

Eixamples

```
[127.0.0.1:4379]> hset empleat_1 nom Andreu
[127.0.0.1:4379]> hset empleat_1 departament
[127.0.0.1:4379]> hset empleat_1 ann 1000.0
[127.0.0.1:4379]>
```

```
[127.0.0.1:4379]> hset empleat_2 nom Berta
[127.0.0.1:4379]> hset empleat_2 ann 1500.0
[127.0.0.1:4379]>
```

HGET

Sintax

```
hget clave campo
```

Torna el valor del camp de la clau. Si no existeix (el camp o la clau) torna nil. Només podem especificar un camp.

Eixamples

```
[127.0.0.1:4379]> hget empleat_1 nom
[127.0.0.1:4379]> hget empleat_1 departament
[127.0.0.1:4379]> hget empleat_2 nom
[127.0.0.1:4379]> hget empleat_2 departament
[127.0.0.1:4379]>
```

HGETALL

Sintax

```
hgetall clave
```

Torna una llista amb tots els camps i els seus valors de la clau. La seqüència és camp1 valor1 camp2 valor2 ... Però no ens podem fer que l'ordre siga el mateix ordre que quan el vam definir.

Eixamples

```
[127.0.0.1:4379]> hgetall empleat_1
[1] "nom"
[2] "departament"
[3] "departament"
[4] "nom"
[5] "1000.0"
```

HDEL

Sintax

```
hdel clave campo1 campo2
```

Elimina el o els camps especificats. Si no existeixen algun d'ells, senzillament ignora i si que elimina els altres.

Eixamples

```
[127.0.0.1:4379]> hdel empleat_1 departament
[127.0.0.1:4379]> hgetall empleat_1
[1] "nom"
[2] "Andreu"
[3] "1000.0"
```

HKEYS

Sintax

```
hkeys clave
```

Torna una llista amb els camps de la clau. Si la clau no existeix, torna una llista buida

Eixamples

```
[127.0.0.1:4379]> hkeys empleat_1
[1] "nom"
```

HVALS

Sintax

```
hvals clave
```

Torna una llista amb els valors (inicisament els valors) de tots els camps de la clau. Si la clau no existeix, torna una llista buida

Eixamples

```
[127.0.0.1:4379]> hvals empleat_1
[1] "Andreu"
[2] "1000.0"
```

Altres Comandos

També existeixen altres comandos, de funcionament com catar a esperar (els hem vist tots en el cas de String):

- **Impat:** Torna mai d'un camp de la clau
- **hsetx:** assigna més d'un camp a una clau
- **hexists:** indica si existeix el subcamp de la clau
- **hincrby:** assigna increment en més de que no existeix el camp.
- **hincrdy:** incrementa el camp de la clau

Les Llistes en Redis són llistes de Strings ordenades, on cada element està associat a un index de la llista. Es poden recuperar els elements tant de forma ordenada (per l'index) com accedint directament a una posició.

Els elements es poden afegir al principi, al final o també en una posició determinada.

La llista es crea en el moment en què s'insereix el primer element, i desapareix quan llueix l'últim element que queda.

Estar molt ben optimitzades per la inserció i per la consulta.

Els comandes que afecten a les llistes comencen quasi tots per L, excepte alguns que comencen per R indicant que fan l'operació per la dreta.

Per cert, els valors dels elements es poden repetir.

L^{PUSH}

Sintaxi

```
lpush clau valor1 valor2 valorN
```

Introdueix els valors a la llista (creant la clau si és necessari). Les insereix en la primera posició, o també podríem dir que per la esquerra (**Left PUSH**), imaginant que els elements estan ordenats d'esquerra a dreta. Si posem més d'un valor, s'aniran introduint sempre en la primera posició. El comandó tornarà el número d'elements (strings) de la llista després de la inserció.

Exemples

```
127.0.0.1:6379> lpush llistal primera segona tercera
(integer) 3
127.0.0.1:6379> lrange llistal 0 -1
1) "tercera"
2) "segona"
3) "primera"
127.0.0.1:6379> lpush llistal quartacincquesa
(integer) 4
127.0.0.1:6379> lrange llistal 0 -1
1) "quintesa"
2) "quarta"
3) "segona"
4) "primera"
5) "cinquena"
```

Nota

Per a veure el contingut de la llista utilitzarem el comandó **lrange llista 0 -1**, que torna la llista sencera. Veurem de forma més completa aquest comandó amb posterioritat.

R^{PUSH}

Sintaxi

```
rpush clau valor1 valor2 valorN
```

Introdueix els valors a la llista (creant la clau si és necessari). Les insereix en l'última posició, o també podríem dir que per la dreta (**Right PUSH**), imaginant que els elements estan ordenats d'esquerra a dreta. El comandó tornarà el número d'elements (strings) de la llista després de la inserció.

Exemples

```
127.0.0.1:6379> rpush llistal sisena setena
(integer) 5
127.0.0.1:6379> lrange llistal 0 -1
1) "setena"
2) "sexta"
3) "quinta"
4) "segona"
5) "primera"
6) "sisena"
7) "setena"
```

L^{POP}

Sintaxi

```
lpop clau
```

Torna el primer element (el de més a la esquerra).

Exemples

```
127.0.0.1:6379> lpop llistal
(integer) 1
127.0.0.1:6379> lrange llistal 0 -1
1) "quinta"
2) "quarta"
3) "segona"
4) "primera"
5) "sisena"
```

R^{POP}

Sintaxi

```
rpop clau
```

Torna el últim element (el de més a la dreta).

Exemples

```
127.0.0.1:6379> rpop llistal
(integer) 1
127.0.0.1:6379> lrange llistal 0 -1
1) "setena"
2) "sexta"
3) "quinta"
4) "segona"
5) "primera"
```

L^{SET}

Sintaxi

```
lset clau index valor
```

Substitueix el valor de la posició indicada per l'index. Tant la clau com l'element de la posició indicada han d'existeixi, sinó donarà error. Ara la L no significa Left sino List.

La primera posició és 0. I també es poden posar números negatius: -1 és l'últim, -2 el penúltim, ...

Exemples

```
127.0.0.1:6379> lset llistal 2 quinta
ok
127.0.0.1:6379> lrange llistal 0 -1
1) "quinta"
2) "quarta"
3) "segona"
4) "primera"
5) "sisena"
6) "setena"
7) "setena"
127.0.0.1:6379> lset llistal -1 cinquena
ok
127.0.0.1:6379> lrange llistal 0 -1
1) "quinta"
2) "quarta"
3) "segona"
4) "primera"
5) "cinquena"
```

Observeu com es poden repartir els valors

L^{INDEX}

Sintaxi

```
lindex clau index
```

Torna l'element situat en la posició indicada per l'index, però sense eliminar-lo de la llista.

Exemples

```
127.0.0.1:6379> lrange llistal 0 -1
1) "quinta"
2) "quarta"
3) "segona"
4) "primera"
5) "sisena"
127.0.0.1:6379> lindex llistal 0
(integer) 1
127.0.0.1:6379> lrange llistal 3
1) "quinta"
2) "quarta"
3) "segona"
4) "primera"
127.0.0.1:6379> lindex llistal -1
(integer) 5
1) "quinta"
2) "quarta"
3) "segona"
4) "primera"
5) "cinquena"
```

L^{INSERT}

Sintaxi

```
linsert clau BEFORE | AFTER valor1 valor2
```

Inserix el valor abans o després (segona i que tindrà) de la primera vegada que troba el valor. No substitueix, sinó que inserix en una determinada posició. Els elements que van després de l'element introduït veuran actualitzat el seu index.

Exemples

```
127.0.0.1:6379> lrange llistal 0 -1
1) "quinta"
2) "quarta"
3) "segona"
4) "primera"
5) "cinquena"
127.0.0.1:6379> linsert llistal BEFORE quarta segona
(integer) 6
127.0.0.1:6379> lrange llistal 0 -1
1) "quinta"
2) "quarta"
3) "segona"
4) "quinta"
5) "primera"
6) "cinquena"
127.0.0.1:6379> linsert llistal BEFORE cinquena sisena
(integer) 7
127.0.0.1:6379> lrange llistal 0 -1
1) "quinta"
2) "quarta"
3) "segona"
4) "quinta"
5) "primera"
6) "cinquena"
7) "sisena"
```

Si intentem inserir abans d'un element que no existeix, tornarà -1 indicant que no l'ha trobat i no farà la inserció.

```
127.0.0.1:6379> linsert llistal BEFORE desena setena
(error) NO EXISTENT
127.0.0.1:6379> lrange llistal 0 -1
1) "quinta"
2) "quarta"
3) "segona"
4) "primera"
5) "cinquena"
6) "sisena"
7) "setena"
```

L^{RANGE}

Sintaxi

```
lrange clau inicio final
```

Torna els elements de la llista inclosos entre els index inici i final, ambos inclosos. El primer element es el 0. Es poden posar valors negatius: -1 és l'últim, -2 el penúltim, ...

Exemples

```
127.0.0.1:6379> lrange llistal 0 -1
1) "quinta"
2) "quarta"
3) "segona"
4) "primera"
5) "cinquena"
127.0.0.1:6379> lrange llistal 2 4
1) "quinta"
2) "quarta"
3) "segona"
4) "primera"
5) "cinquena"
127.0.0.1:6379> lrange llistal 1 -2
1) "quinta"
2) "quarta"
3) "segona"
4) "primera"
5) "cinquena"
127.0.0.1:6379> lrange llistal 4 4
1) "primera"
```

L^{LLEN}

Sintaxi

```
llen clau
```

Torna el número d'elements de la llista

Exemples

```
[127.0.0.1:4070]> llen listal
(integer) 7
```

LREM

Sintax

llen cleo numero valor

Elimina els elements de la llista que coincideixen amb el valor proporcionat. Ja sabem que els valors es poden repetir. Amb el número indiquem quants elements volem que s'eliminen començant per l'esquerra: si posem 1 s'eliminara el primer element amb aquest valor, si posem 2 s'eliminaran els dos primers elements (els de més a l'esquerra) que tinguin aquest valor. Si posem 0 s'eliminaran tots els elements amb aquest valor.

Exemples

```
[127.0.0.1:4070]> rpush listal segona
[127.0.0.1:4070]> llen listal 0 -1
1) "quarta"
2) "tercera"
3) "segona"
4) "quarta"
5) "tercera"
6) "segona"
7) "quarta"
8) "tercera"
9) "segona"
```

```
[127.0.0.1:4070]> llen listal 1 segona
[127.0.0.1:4070]> llen listal 0 -1
1) "quarta"
2) "tercera"
3) "segona"
4) "quarta"
5) "tercera"
6) "segona"
7) "quarta"
8) "tercera"
9) "segona"
```

```
[127.0.0.1:4070]> llen listal 5 quarta
[127.0.0.1:4070]> llen listal 0 -1
1) "tercera"
2) "segona"
3) "quarta"
4) "quarta"
5) "quarta"
```

LTRIM

Sintax

ltrim cleo index final

Elimina els elements que quedan fora dels índex inici i final, és a dir elimina els que estiguin a l'esquerra d'inici, i els que estiguin a la dreta de final.

Exemples

```
[127.0.0.1:4070]> ltrim listal 1 -2
[127.0.0.1:4070]> llen listal 0 -1
1) "primera"
2) "segona"
3) "tercera"
```

Els **Set** de Redis són conjunts de valors de tipus String no ordenats. Podrem afegeix, actualitzar i elaborar aquests elements de forma clàmida i eficient. No es permeten els valors duplicats.

A més Redis ens ofereix operacions interessants com la unió, intersecció i diferència de conjunts.

Com sempre, els comandos són específics, és a dir no era valer els de String, List o Hash. Tots els comandos comencen per **S**.

SADD

Sintax

```
sadd clave valor1 valor2 valorN
```

Afegeix els valors al conjunt (creant la clau si és necessari). Recordeu que l'ordre no és important, i que no es poden repetir els valors; si intentem introduir un repitit, no donarà error, però no s'introduirà. El comand tornarà el número d'elements que realment s'han afegeix.

Exemples

```
127.0.0.1:4379> sadd colores colg verd blau
(integer) 3
127.0.0.1:4379> sadd colores verd groc
(integer) 4
```

SMEMBERS

Sintax

```
smembers clave
```

Torna tots els valors del conjunt. Si la clau no existeix tornarà un conjunt buit. Recordeu que l'ordre dels elements no és predefinit.

Exemples

```
127.0.0.1:4379> smembers colores
[1] "groc"
[2] "blau"
[3] "verd"
[4] "negre"
```

SISMEMBER

Sintax

```
sismembers clave valor
```

Comprova si el valor està en el conjunt, tornant 1 en cas afirmatiu i 0 en cas negatiu.

Exemples

```
127.0.0.1:4379> sismembers colores verd
(integer) 1
127.0.0.1:4379> sismembers colores negre
(integer) 0
```

SCARD

Sintax

```
scard clave
```

Torna la cardinalitat, és a dir, el nombre d'elements del conjunt en l'actualitat.

Exemples

```
127.0.0.1:4379> scard colores
(integer) 4
```

SREM

Sintax

```
srem clave valor1 valor2 valorN
```

Elimina els valors del conjunt. Si el conjunt es queda buit, eliminara la clau també. Si algun dels valor no es cap element del conjunt, senzillament s'ignora. El comand torna el nombre d'elements realment eliminat.

Exemples

```
127.0.0.1:4379> srem colores verd negre
(integer) 2
127.0.0.1:4379> smembers colores
[1] "blau"
[2] "roig"
[3] "verd"
```

SPOP

Sintax

```
spop clave
```

Torna el primer element aleatori del conjunt. Recordeu que a més de tornar-lo, l'elimina del conjunt.

Exemples

```
127.0.0.1:4379> smembers colores
[1] "roig"
[2] "blau"
[3] "verd"
[4] "negre"
127.0.0.1:4379> spop colores
[1] "roig"
127.0.0.1:4379> smembers colores
[2] "blau"
[3] "verd"
[4] "negre"
```

SRANDMEMBER

Sintax

```
srandommember clave
```

Molt paregut a l'anterior. Torna un valor aleatori del conjunt, però en aquesta ocasió no l'elimina del conjunt.

Exemples

```
127.0.0.1:4379> srandommember colores
[1] "blau"
[2] "roig"
[3] "verd"
[4] "negre"
```

SUNION

Sintax

```
union clave1 clave2 claveN
```

Torna la unió dels elements dels conjunts especificats. És una unió correcta, és a dir, no es repeteix cap valor.

No modifica cap conjunt, i el resultat únicament es torna, no es guarda en cap lloc de forma permanent.

Exemples

```
127.0.0.1:4379> smembers colores
[1] "roig"
[2] "blau"
[3] "verd"
[4] "negre"
127.0.0.1:4379> sadd colores1 verd roig groc
(integer) 3
127.0.0.1:4379> smembers colores1
[1] "roig"
[2] "blau"
[3] "verd"
[4] "negre"
127.0.0.1:4379> sunion colores colores1
[1] "roig"
[2] "blau"
[3] "verd"
[4] "negre"
[5] "groc"
127.0.0.1:4379> srem colores1 groc
(integer) 1
127.0.0.1:4379> smembers colores1
[1] "roig"
[2] "blau"
[3] "verd"
[4] "negre"
```

SUNIONSTORE

Sintax

```
unionstore clave_desti clave1 clave2 claveN
```

Igual que l'anterior, però ara si que es guarda el resultat de la unió en un conjunt, **clave_desti** (el primer especificat). Si la **clave_desti** ja existia, substituirà el contingut.

Exemples

```
127.0.0.1:4379> smembers colores
[1] "roig"
[2] "blau"
[3] "verd"
[4] "negre"
127.0.0.1:4379> smembers colores1
[1] "roig"
[2] "blau"
[3] "verd"
[4] "negre"
127.0.0.1:4379> sadd colores1 groc
(integer) 1
127.0.0.1:4379> sunionstore colores2 colores colores1
[1] "roig"
[2] "blau"
[3] "verd"
[4] "negre"
[5] "groc"
127.0.0.1:4379> smembers colores2
[1] "roig"
[2] "blau"
[3] "verd"
[4] "negre"
[5] "groc"
```

SDIFF

Sintax

```
sdiff clave1 clave2 claveN
```

Torna la diferència dels elements del primer conjunt respecte de la unió de tots els altres. És a dir, torna els elements del primer conjunt que no pertanyen acap dels altres.

No modifica cap conjunt, i el resultat únicament es torna, no es guarda en cap lloc de forma permanent.

Exemples

```
127.0.0.1:4379> smembers colores
[1] "roig"
[2] "blau"
[3] "verd"
[4] "negre"
127.0.0.1:4379> smembers colores1
[1] "roig"
[2] "blau"
[3] "verd"
[4] "negre"
127.0.0.1:4379> sdiff colores1 colores
(integer) 0
127.0.0.1:4379> smembers colores1
[1] "roig"
[2] "blau"
[3] "verd"
[4] "negre"
```

SDIFFSTORE

Sintax

```
sdiffstore clave_desti clave1 clave2 claveN
```

Igual que l'anterior, però si que es guarda el resultat de la diferència en un conjunt, **clave_desti** (el primer especificat). Si la **clave_desti** ja existia, substituirà el contingut.

Exemples

```
127.0.0.1:4379> smembers colores
[1] "roig"
[2] "blau"
[3] "verd"
[4] "negre"
127.0.0.1:4379> smembers colores1
[1] "roig"
[2] "blau"
[3] "verd"
[4] "negre"
127.0.0.1:4379> sadd colores1 groc
(integer) 1
127.0.0.1:4379> sdiffstore colores3 colores1 colores
(integer) 0
127.0.0.1:4379> smembers colores3
[1] "roig"
[2] "blau"
[3] "verd"
[4] "negre"
```

SINTER

Sintax

```
sinter clave1 clave2 claveN
```

Torna la intersecció dels elements dels conjunts. És a dir, torna els elements que pertanyen a tots els conjunts especificats.

No modifica cap conjunt, i el resultat únicament es torna, no es guarda en cap lloc de forma permanent.

Exemples

```
127.0.0.1:4379> smembers colores
[1] "roig"
[2] "blau"
[3] "verd"
[4] "negre"
127.0.0.1:4379> smembers colores1
[1] "roig"
[2] "blau"
[3] "verd"
[4] "negre"
127.0.0.1:4379> sinter colores colores1
[1] "roig"
[2] "blau"
[3] "verd"
[4] "negre"
```

SINTERSTORE

Sintax

```
sinterstore clave_desti clave1 clave2 claveN
```

Igual que l'anterior, però si que es guarda el resultat de la intersecció en un conjunt, **clave_desti** (el primer especificat). Si la **clave_desti** ja existia, substituirà el contingut.

Exemples

```
127.0.0.1:4379> smembers colores
[1] "roig"
[2] "blau"
[3] "verd"
[4] "negre"
127.0.0.1:4379> smembers colores1
[1] "roig"
[2] "blau"
[3] "verd"
[4] "negre"
127.0.0.1:4379> sinterstore colores4 colores colores1
(integer) 0
127.0.0.1:4379> smembers colores4
[1] "roig"
[2] "blau"
[3] "verd"
[4] "negre"
```

```
sinteriorc clau_desti clau1 clau2 clau3
```

Igual que l'anterior, però ara si que es guarda el resultat de la intersecció en un conjunt, clau_desti (el primer especificat). Si la clau_desti ja existia, substituirà el contingut.

Exemples

```
127.0.0.1:4270> smembers colors
1) "blau"
2) "blau"
127.0.0.1:4270> smembers coloral
1) "grau"
2) "verd"
3) "roig"
4) "verd"
127.0.0.1:4270> sinteriorc coloral colors
(integer) 1
127.0.0.1:4270> smembers colors4
1) "roig"
```

SMOVE

Sintax

```
move clau_origen clau_desti valor
```

Moure el valor del conjunt origen (el primer conjunt) al conjunt destí (el segon). Això suposarà eliminar-lo del primer i afegir-lo al segon. Tornarà 1 si s'ha menejat, i 0 si no l'ha menejat.

Exemples

```
127.0.0.1:4270> smembers colors
1) "verd"
2) "blau"
127.0.0.1:4270> smembers coloral
1) "grau"
2) "verd"
3) "roig"
4) "verd"
127.0.0.1:4270> smove coloral colors verd
(integer) 1
127.0.0.1:4270> smembers colors
1) "roig"
2) "blau"
3) "grau"
4) "verd"
127.0.0.1:4270> smembers coloral
1) "roig"
2) "blau"
```

3.2.6 - Set ordenat

Els Sets ordenats (Sorted Set) de Redis són Sets que a més de guardar els valors, guarden també una puntuació (score) per a cada valor. Redis mantindrà el conjunt ordenat per aquesta puntuació.

Els valors no es podran repetir, però si les puntuacions.

Molts dels comandos seran iguals que els del Set, ja que un conjunt ordenat no deixa de ser un conjunt, però amb la informació de la puntuació. En aquesta ocasió començaran per Z.

ZADD

Sintax
`zadd clave puntuacion1 valor1 puntuacion2 valor2 ... puntuacionN valorN`

Afegeix els valors al conjunt (creant la clau si és necessari) amb les puntuacions corresponents. Les puntuacions seran Strings de valors reals (float). No es poden repetir els valors, però sí les puntuacions. Si intentem introduir un valor repetit, el qual farà ser actualitzar la puntuació. El comando tornarà el nombre d'elements que realment s'han afegit.

Exemples

```
[127.0.0.1:6379> zadd puntuacion 1 Nom1 2 Nom2 5 Nom3 4 Nom4
(integer) 4
[127.0.0.1:6379> zrange puntuacion 0 -1
1) "Nom1"
2) "Nom2"
3) "Nom3"
4) "Nom4"
```

ZCARD

Sintax
`zcard clave`

Torna la cardinalitat, és a dir, el nombre d'elements del conjunt ordenat en l'actualitat.

Exemples

```
[127.0.0.1:6379> zcard puntuacion
(integer) 4
```

ZSCORE

Sintax
`zscore clave valor`

Torna la puntuació (score) del valor específic del conjunt ordenat. Si no existeix el valor o no existeix la clau, torna nil.

Exemples

```
[127.0.0.1:6379> zscore puntuacion Nom1
1)
[127.0.0.1:6379> zscore puntuacion Nom7
nil]
```

ZCOUNT

Sintax
`zcount clave min max`

Torna el nombre de valors que estan entre les puntuacions especificades (ambdós inclosos).

Exemples

```
[127.0.0.1:6379> zcount puntuacion 2 5
(integer) 4
```

ZRANGE

Sintax
`zrange clave inicio final [withscores]`

Torna els elements del conjunt ordenat que tenen una puntuació compresa entre els índexs inici i final, ambdós inclosos. I es trauen per ordre ascendent de puntuació. El primer element és el 0. Es poden posar valors negatius, sent -1 l'últim, -2 el penúltim,... Opcionalment podem posar WITHSCORES per a que ens torni també la puntuació de cada element.

Exemples

```
[127.0.0.1:6379> zrange puntuacion 0 -1
1) "Nom1"
2) "Nom2"
3) "Nom3"
4) "Nom4"
5) "Nom5"
6) "Nom6"
7) "Nom7"
8) "Nom8"
9) "Nom9"
10) "Nom10"

Si voleu que el conjunt en ordre invers de puntuació, utilitzareu el comando ZREVRANGE (reverse range).
[127.0.0.1:6379> zrevrange puntuacion 0 -1 withscores
1) "Nom10"
2) "Nom9"
3) "Nom8"
4) "Nom7"
5) "Nom6"
6) "Nom5"
7) "Nom4"
8) "Nom3"
9) "Nom2"
10) "Nom1"
```

ZRANGEBYSCORE

Sintax
`zrangebyscore clave min max [withscores]`

Torna els elements del conjunt ordenat que tenen una puntuació compresa entre min i max (ambdós inclosos). I es trauen per ordre ascendent de puntuació. Opcionalment podem posar WITHSCORES per a que ens torni també la puntuació de cada element.

Exemples

```
[127.0.0.1:6379> zrangebyscore puntuacion 2 5
1) "Nom1"
2) "Nom2"
3) "Nom3"
4) "Nom4"
5) "Nom5"
6) "Nom6"
7) "Nom7"
8) "Nom8"
9) "Nom9"
10) "Nom10"

Si voleu que les puntuacions foren estricteument majors que la puntuació mínima i/o estrictelement menor que la puntuació màxima, posareu un parèntesi davant de min i/o max.
[127.0.0.1:6379> zrangebyscore puntuacion 2 0 withscores
1) "Nom1"
2) "Nom2"
3) "Nom3"
4) "Nom4"
5) "Nom5"
6) "Nom6"
7) "Nom7"
8) "Nom8"
9) "Nom9"
10) "Nom10"

I si voleu traure un conjunt en ordre invers de puntuació, utilitzareu el comando ZREVRANGEBYSCORE (reverse range). Cuideu que com ve en ordre invers, ara el valor màxim ha de ser el primer i el mínim el segon.
[127.0.0.1:6379> zrevrangebyscore puntuacion 5 2 withscores
1) "Nom10"
2) "Nom9"
3) "Nom8"
4) "Nom7"
5) "Nom6"
6) "Nom5"
7) "Nom4"
8) "Nom3"
9) "Nom2"
10) "Nom1"
```

ZRANK

Sintax
`zrank clave valor`

Torna el nombre d'ordre de l'element amb el valor específic. El primer valor de el 0. Si no existeix, torna nil.

Exemples

```
[127.0.0.1:6379> zrank puntuacion Nom1
(integer) 0
[127.0.0.1:6379> zrank puntuacion Nom4
(integer) 3
[127.0.0.1:6379> zrank puntuacion Nom7
(integer) 6

Si voleu saber el nombre d'ordre però des del final de la llista (en ordre invers), hem d'utilitzar ZREV RANK.
```

```
[127.0.0.1:6379> zrevrank puntuacion Nom1
(integer) 9
[127.0.0.1:6379> zrevrank puntuacion Nom4
(integer) 6
[127.0.0.1:6379> zrevrank puntuacion Nom7
(integer) 3
```

ZREM

Sintax
`zrem clave valor1 valor2 ... valorN`

Elimina els elements amb els valors especificats. Si algun valor no existeix, senzillament l'ignora. Torna el nombre d'elements realment eliminats.

Exemples

```
[127.0.0.1:6379> zrem puntuacion Nom1
(integer) 1
[127.0.0.1:6379> zrem puntuacion 0 -1 withscores
1) "Nom1"
2) "Nom2"
3) "Nom3"
4) "Nom4"
5) "Nom5"
6) "Nom6"
7) "Nom7"
8) "Nom8"
9) "Nom9"
10) "Nom10"
```

ZREMRANGEBYSCORE

Sintax
`zremrangebyscore clave min max`

Elimina els elements amb puntuació compresa entre el mínim i el màxim de forma inclusiva els valors especificats. Si voleu fer-ho de forma exclusiva (sense incloure les puntuacions dels extrems) posareu un parèntesi davant del mínim i/o el màxim. Torna el nombre d'elements realment eliminats.

Exemples

```
[127.0.0.1:6379> zremrangebyscore puntuacion 1 4
(integer) 3
[127.0.0.1:6379> zrange puntuacion 0 -1 withscores
1) "Nom1"
2) "Nom2"
3) "Nom3"
4) "Nom4"
5) "Nom5"
6) "Nom6"
```

ZINCRBY

Sintax
`zincrby clave increment value`

Incremen la puntuació de l'element especificat. El valor de la puntuació a incrementar és un número real. Torna el valor la puntuació final de l'element. Si l'element no existeix, l'insereix, assumint una puntuació inicial de 0.

Exemples

```
[127.0.0.1:6379> zincrby puntuacion 1.5 Nom2
1)
[127.0.0.1:6379> zincrby puntuacion 2.75 Nom5
2)
[127.0.0.1:6379> zrange puntuacion 0 -1 withscores
1) "Nom2"
2) "Nom5"
3) "Nom7"
4) "Nom9"
5) "Nom10"
6) "Nom1"
```

La utilització des de Java és molt senzilla, i podem utilitzar tots els comandos que hem vist.

Haurem d'incorporar en el projecte una llibreria. Podem utilitzar per exemple la de **Jedis** (acrònim de Java Redis). A la següent adreça la podreu trobar (no és l'última versió, però ens val):

<http://search.maven.org/remotecontent?filepath=redis/jedis/jedis/2.0.0/jedis-2.0.0.jar>

Per a fer proves, creem un nou projecte anomenat **Tema7_2**, i incorporen la llibreria de **Jedis**. Creem un paquet anomenat **Exemples**, per separar aquests dels exercicis.

Connexió

La connexió és tan senzilla com el següent:

```
Jedis con = new Jedis("localhost");
```

És a dir, obtenim un objecte **Jedis** passant al constructor l'adreça del servidor, i després connectem amb el mètode **connect**.

Si el servidor no es tenia en la mateixa màquina, només haurem de substituir **localhost** per l'adreça o estació del servidor.

L'objecte **Jedis** representarà una connexió amb el servidor **Redis**. Tots els comandos que hem vist per a utilitzar des del client de **Redis**, seran mètodes d'aquest objecte. Només haurem d'anar amb compte amb el que ens torna el servidor quan fem una petició. Moltes vegades serà un **String**, però en moltes altres ocasions seran col·leccions: **sets**, **lists**...

Per a llocar la connexió:

```
con.close();
```

Comandos que tornen Strings

Tots els comandos que van veure seran els mètodes de l'objecte **Jedis**. En concret obtindrem el valor d'una clau (comando **get clau**) serà el mètode **get** al qual li passarem la clau com a paràmetre:

Aci tenim ja la primera prova:

```
import redis.clients.jedis.Jedis;
public class Prova {
    public static void main(String[] args) {
        Jedis con = new Jedis("localhost");
        con.connect();
        System.out.println(con.get("value"));
        con.close();
    }
}
```

Si volem guardar una clau amb un valor, utilitzarem el mètode **set(clau,valor)**, al qual com veiem li hem de passar els dos paràmetres:

```
import redis.clients.jedis.Jedis;
public class Prova {
    public static void main(String[] args) {
        Jedis con = new Jedis("localhost");
        con.connect();
        String value = "Aquesta clau es una clau creada des de Java";
        con.set("clau_Java", value);
        System.out.println("clau_Java");
        con.close();
    }
}
```

Comandos que tornen conjunts

En moltes ocasions, si ens torna **Redis** no és un **String**, sinó un conjunt de **String**. Així serà el que trobarem amb **sets**, però també en moltes altres ocasions. Per exemple quan demanem uns quants claus amb **MGET**, o quan utilitzem **KEYS** per a que ens torni les claus que coincideixin amb el patró.

En algunes ocasions si haurem d'emplegar amb un objecte **List** que no impedeix ordre. En altres ocasions amb un objecte **List** quan apostem ordre si que impedeix.

Per exemple, si volem obtenir els valors d'unes quantes claus, si que importa l'ordre (el primer valor es de la primera clau, el segon de la segona). Aleshores ho obtindrem en un **List**:

```
import java.util.List;
import redis.clients.jedis.Jedis;
public class Prova {
    public static void main(String[] args) {
        Jedis con = new Jedis("localhost");
        con.connect();
        List<String> list = con.keys("me*");
        for (String s : list)
            System.out.println(s);
        con.close();
    }
}
```

Però en canvi si volem obtenir totes les claus utilitzarem el mètode **keys** passant-li el patró com a paràmetre. L'ordre no importa i a més no es poden predir. Per tant hem d'emplegar el resultat amb un **Set**:

```
import java.util.Set;
import redis.clients.jedis.Jedis;
public class Prova {
    public static void main(String[] args) {
        Jedis con = new Jedis("localhost");
        con.connect();
        Set<String> set = con.keys("*");
        for (String s : set)
            System.out.println(s);
        con.close();
    }
}
```

I evidentment també es pot fer que accidiem amb el tipus de dades **List**, **Set** i segurament també **Hash**.

Per a accedir a tot el contingut d'un **Set** utilitzem per exemple **range 0 -1**. Si utilitzem aquest mètode de **Jedis** ens tornarà un **List** (de Java):

```
import java.util.List;
import redis.clients.jedis.Jedis;
public class Prova {
    public static void main(String[] args) {
        Jedis con = new Jedis("localhost");
        con.connect();
        List<String> ll = con.range("llista1", 0, -1);
        for (String s : ll)
            System.out.println(s);
        con.close();
    }
}
```

Sí es un **Set** accedirem amb el mètode **smembers** que tornarà un **Set** (de Java):

```
import java.util.Set;
import redis.clients.jedis.Jedis;
public class Prova {
    public static void main(String[] args) {
        Jedis con = new Jedis("localhost");
        con.connect();
        Set<String> a = con.smembers("collecc");
        for (String s : a)
            System.out.println(s);
        con.close();
    }
}
```

I en el cas de **Hash**, amb el mètode **hkeys** podem obtenir tots els camps (subclaus), i a partir d'ells els seus valors. El que torna **hkeys** és un **Set** (de Java):

```
import java.util.Set;
import redis.clients.jedis.Jedis;
public class Prova {
    public static void main(String[] args) {
        Jedis con = new Jedis("localhost");
        con.connect();
        Set<String> subcamp = con.hkeys("empleat_1");
        for (String s : subcamp)
            System.out.println(subcamp + " : " + con.hget("empleat_1", s));
        con.close();
    }
}
```

Tractament dels conjunts ordenats

Els conjunts ordenats (**Sorted Sets**) tenen mètodes específics, igual que tots els tipus. Algunes d'aquests mètodes tornen **String**s, i uns altres conjunts (**Set** de Java). No hi ha problema amb aquests tipus, que ja els hem tractat.

Pot haver-hi una mica més que tornar tots els valors amb més d'un valor per cada element. Es el cas dels mètodes **ZRANGE** (amb les variants **ZREVRANGE**, **ZRANGEBYSCORE** i **ZREVRANGEBYSCORE**), que tenen la possibilitat de fer el paràmetre **WITHSCORES**. En aquest cas cada element constarà del valor i de la puntuació. El que tornaran els mètodes és un **Set** de **Tuples**, objecte proporcionat per **Jedis**, que disposarà dels mètodes **getElement** per al valor i **getScore** per a la puntuació:

```
import java.util.Set;
import redis.clients.jedis.Tuple;
public class Prova {
    public static void main(String[] args) {
        Jedis con = new Jedis("localhost");
        con.connect();
        Set<Tuple> conJsd = con.zrangeWithScores("puntuaciones", 0, -1);
        for (Tuple t : conJsd)
            System.out.println(t.getElement() + " --> " + t.getScore());
        con.close();
    }
}
```

Segurament MongoDB és el més famós dels Sistemes Gestors de Bases de Dades NoSQL.

El nom de **MongoDB** prové de la paraula anglesa **humongous**, que significa enorme, que és el propòsit d'aquesta Base de Dades: guardar grans quantitats d'informació. És de codi obert i està programada en C++. El va crear l'empresa **10gen** (actualment **MongoDB Inc.**)

És un servidorMongo poten haver més d'una Base de Dades (encara que no sols hi ha una: **test**).

• En cada Base de Dades la informació es guarda en **coleccions**.

• Cada colecció té documents.

• Cada document seran una sèrie de clau-valor, dels tipus suportats per MongoDB, i amb el format JSON (en realitat BSON).

Per tant, en MongoDB hi ha taules. Mirem uns exemples de documents (JSON) per a guardar la informació de llibres i autors. Dependrà de com volem d'accèsser a la informació ens podrem plantear guardar els llibres amb els seus autors, o guardar els autors, amb els seus llibres. Fins i tot ens podríem guardar els dos, per a poder accedir de totes les maneres, encara que és a costa de doblar la informació.

De la primera manera, guardant els llibres amb el seu autor, podríem tenir documents amb aquesta estructura, que es podrien guardar en una col·lecció anomenada **Llibres**:

```
[{"_id":101,
  "titol":"El secret de Khadeall",
  "autor": {
    "nom": "Pep",
    "cognoms": "Castellano Puchol",
    "any_naixement": 1960,
    "lloc_naixement": "Barcelona",
    "lloc_residència": "Barcelona",
    "telefon": "+34-95420-72-3"
  }
}, {"_id":102,
  "titol": "Ombra del Vent",
  "autor": {
    "nom": "Carles",
    "cognoms": "Santos Safo",
    "any_naixement": 1950,
    "lloc_naixement": "Reus",
    "lloc_residència": "Barcelona",
    "telefon": "+34-95420-72-3"
  }
}]
```

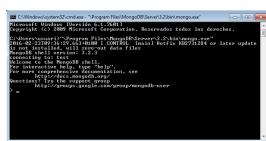
Observau com els objectes no tenen per què accedir al mateix estructura. La manera d'accèsser al nom d'un autor seria aquesta: `objecte.autor.nom`

Una manera alternativa de guardar la informació, com havíem comentat abans seria organitzar per autors, amb els seus llibres. D'aquesta manera podríem anar omplint la col·lecció **Autors** amb un o més documents d'aquest estil:

```
[{"_id": 201,
  "nom": "Pep",
  "cognoms": "Castellano Puchol",
  "any_naixement": 1960,
  "llocs": [
    {
      "titol": "El secret de Khadeall",
      "any": 2000,
      "lloc": "+34-95420-72-3"
    },
    {
      "titol": "Mabatgeais 102",
      "any": 2001,
      "lloc": "+34-95420-72-3"
    }
  ]
}, {"_id": 202,
  "nom": "Carles",
  "cognoms": "Santos Safo",
  "any_naixement": 1950,
  "llocs": [
    {
      "titol": "Ombra del Vent",
      "any": 2002,
      "lloc": "+34-95420-72-3"
    }
  ]
}]
```

Observeu com per a un autor, ja tenim un array (els claudàtors []) amb els seus llibres.

Quina de les dues maneres és millor per a guardar la informació? Doncs depèn de l'accés que s'haja de fer a les dades. La millor serà segurament aquella que depèn dels accessos que s'hagen de fer, torna la informació de forma més ràpida.



Per a provar el seu funcionament, anem a fer un parell de comandes: un per a guardar un document i un altre per a recularer-lo.

Per a qualsevol operació s'ha de posar `db` seguit del nom de la col·lecció, i després l'operació que volem fer. Amb el següent:

```
[db.example.insert({name:'Bob', que:'ta5'})]
```

Era contestarà:

```
WriteResult({ n: 1 })
```

Indicant que ha inserit un document en la col·lecció `example` (si no estava creada, la crearà).

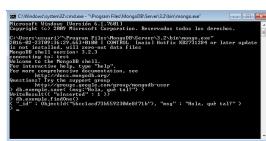
I amb el següent comando recuperarem la informació:

```
[db.example.find()}
```

Que ens tornarà:

```
[{"_id": "5c4511cd1d755923505a7f13"}, "name": "Bob", que: "ta5"}]
```

Tot ho fa en la mateixa terminal, i a cadaçó de nosaltres ens donarà un número diferent en `_Objectid`. En la següent imatge es veuen les dues operacions:



En realitat estem connectats a una Base de Dades anomenada `test`. Podem crear i utilitzar més d'una Base de Dades, però en aquest curs tindrem més que suficient amb aquesta Base de Dades. Per a comprovar-ho podem executar la següent sentència, que ens torna el nom de la Base de Dades:

```
[db.getCollection()
```

```
test
```

Començarem la utilització de MongoDB des de la consola que havíem arrancat al final de la instal·lació.

Recordeu que s'abreix la terminal:

- Una altra al servidor en mode ú (que no hem de lancer) : mongod
- Una altra al client que es connecta al servidor : mongo

En aquesta última consola del client podem utilitzar sentències del llenguatge Javascript, però el que més ens interessarà, evidentment, són les sentències d'accés a dades. Del llenguatge Javascript pràcticament tinc que utilitzarem són variables i algunes funcions.

Utilització de variables

Com començarem el que més utilitzarem del llenguatge Javascript és la utilització de variables, que ens pot ser molt útil en algunes ocasions. Podrem utilitzar-les durant la sessió, però evidentment no perduraran d'una sessió a l'altra.

Per a definir una variable podem posar opcionalment davant la paraula reservada var, però no és necessari. Posarem el nom de la variable, el signe igual, i a continuació el valor de la variable, que pot ser una constant, o una expressió utilitzant constants, operadors, altres variables, funcions de Javascript, ...

Especialment interessant són les variables que poden contenir un document JSON.

Per exemple:

```
1 x = "30"
2 y = 4
3 z = 5
4
5 var campObj
6 campObj = {
7   docId: "538462797025935d",
8   docName: "Bilba",
9   camp1: "Bilba",
10  camp2: 45,
11  camp3: new Date()
12 }
13
14 campObj["camp1"] = "Bilbao";
15 campObj["camp2"] = 45;
16 campObj["camp3"] = ISODate("2014-02-23T19:10:29.562Z")
```

Una variable de tipus JSON es podrà modificar molt fàcilment, tota ella, o algun dels elements. Per a aniar als elements posarem nom_variable.nom_camp:

```
1 doc.camp4 = "3.141592"
2
3 doc.camp4 = [ 2 , 4 , 6 , 8 ]
```

I si ara intentem traure el contingut de la variable:

```
1 doc
2 {
3   "camp1": "Bilba",
4   "camp2": 45,
5   "camp3": ISODate("2014-02-24T22:31:12.724Z"),
6   "camp4": [
7     2,
8     4,
9     6,
10    8
11  ]
```

També hem de fer constar que en un document, que s'està de tipus JSON (pràcticament), serà un conjunt de paraules clau-valor, amb algunes restriccions:

- El document (que mostre regalades l'associarem a objecte de JSON) va entre claus {}
- Les claus han de ser strings, i els valors han de ser tipus JSON (o altres tipus).
- La clau no pot ser nulla, ni repetir-se en el mateix objecte (si en diferents objectes, clair)

• Els valors són del tipus que volem en la pregunta 3.2.2

• Un document guardat ha de contenir obligatoriament un camp anomenat _id i que contindrà un valor únic en la col·lecció i servirà per a identificar-lo. Si en guardar un document no li hem posat camp _id, el generarà automàticament MongoDB.

4.2.1 - Tipus de dades

Els valors dels elements, és a dir de les parelles clau valor, poden ser d'una amplia gamma de tipus. Fem un ràpid repàs.

En els exemples que van continuació definim senzillament parelles clau-valor dels diferents tipus, o en tot cas ens ho guardem en variables, però no guardarem encara en la Base de Dades (no farem en la següent pregunta).

Quan guardem en una variable es mostrarà el prompt, la definició de la variable i després el resultat d'haver guardat la variable. Utilitzarem requerdies blancs. Els requerdies grocs són únicament de la definició d'una clau-valor d'un determinat tipus.

NULL

Més que un tipus de dades és un valor, més dit, l'absència de valor

```
[ "x": null ]
```

BOOLEAN

El tipus boolean, que pot agafar els valors true o false.

```
[ "x": true ]
```

```
[ "y": false ]
```

NUMBER

Per defecte, el tipus de dades numèrica serà el de coma flotant (float), simple precisió. Si volem un altre tipus (enter, doble precisió, ...) ho hauríem d'indicar expressament. Així els dos següents valors són float:

```
[ "x": 3.14 ]
```

```
[ "y": 3 ]
```

Si volem que sigui exactament enter, per exemple, hauríem d'utilitzar una funció de conversió:

```
[ "x": Number.parseInt("3.14") ]
```

```
[ "y": Number.parseInt("3") ]
```

STRING

Es pot guardar qualsevol cadena amb caràcters de la codificació UTF-8

```
[ "a": "Hola, que t'as?" ]
```

DATE

Es guarda data i hora, i intentarem es guardar en mil·segons des de l'any inicial. No es guarda el Time zone, és a dir, la desviació respecte a l'hora internacional.

```
[ "x": ISODate("2014-02-24T11:15:27.471Z") ]
```

Normalment utilitzarem fonctions de tractament de la data-hora. L'anterior era per a convertir el string en data-hora. La següent és per a obtenir la data-hora actual:

```
[ "x": new Date() ]
```

És a dir, que si no posem parametre, ens dóna la data-hora actual. Però si podem posar com a parametre la data-hora que volem que genere. En aquest exemple, només posem data, per tant l'hora serà les 00:00:

```
[ "x": new Date("2014-02-21T00:00:00Z") ]
```

En aquest si que posem una determinada hora, i observem com hem deposat la T (Time) entre el dia i el hora:

```
[ "x": new Date("2014-02-21T00:00:00Z") ]
```

```
[ "x": new Date("2014-02-21T00:00:00Z") ]
```

És molt important que posem sempre New Date() per a generar una data-hora. Si possem únicament Date(), el que estem generant és un string (separant amb la data i hora actual, però un string):

```
[ "x": Date("2014-02-21T00:00:00Z") ]
```

```
[ Fri Feb 26 2014 08:30:13 GMT+0100 (CET) ]
```

ARRAY

Es un conjunt d'elements, cadaçun de qualsevol tipus, encara que el més habitual és que siguin del mateix tipus. Van entre claudadors ([]) i els elements separats per comas.

```
[ [ 1, 2, 3, 4, 5, 6 ] ]
```

Com conservarem, cada element de l'array pot ser de qualsevol tipus

```
[ [ y: 1, 2, 3, 14, "Hola", new Date() ] ]
```

En MongoDB podem treballar molt bé amb arrays, i hi hem operacions per a poder buscar dins de l'array, modificar un element, crear index, ...

DOCUMENTS (OBJECTES)

Els documents poden contenir com a elements uns altres documents (**objects** en la terminologia JSON, però **documents** en la terminologia de MongoDB).

Van entre claus (:) i els elements que contindran van separats per comas i seran parelles clau-valor de qualsevol tipus (fins i tot altres documents).

```
[ { a: 1, b: 2 } ]
```

Posar documents dins d'altres documents (el que s'anomena **embedded document**) ens permet guardar la informació d'una manera més real, no tan plana. Així per exemple, les dades d'una persona les podríem definir de la següent manera. Les posarem en una variable, per veure després com podem accedir als diferents elements, encara que el més normal seria guardar-lo en la Base de Dades (amb **insert()** o **save()**). Si copiem el que va a continuació al terminal de MongoDB, ens apareixerà un format estrany. És perquè la sentència d'assiganció a la variable ocupa més d'una línia, i apareixeran 3 punts al principi per a indicar que continua la sentència. Però funcionarà perfectament:

```
doc = {  
    nom: "Juan Martí",  
    adreça: {  
        carrer: "Majors",  
        número: 12,  
        localitat: "Castelldefels"  
    },  
    telèfon: 964223344, 673345123  
}
```

Observau com aquesta estructura que ha quedat tan clara, segurament en una Base de Dades Relacional ens hauria tocat guardar en 3 taules: la de persones, la d'adreces i la de telèfons.

Per a accedir als elements d'un document posarem el punt. Doncs el mateix per als elements d'un document dins d'un document. I també podem accedir als elements d'un array, posant l'índex entre claudadors.

```
[ doc.  
  doc.adreça.  
  doc.adreça.carrer  
  doc.adreça.número  
  doc.adreça.localitat  
  doc.telèfon ]
```

És un tipus que defineix MongoDB per a poder obtenir valors únics. Es el valor per defecte de l'element **_id**, necessari en tot document (atenció: en un document, no en un element de tipus document que hem dit equivalent a objecte de JSON). Es un número long, és a dir que utilitzarà 24 bytes.

Farem proves de la seua utilització en la següent pregunta, en el moment d'inserir diferents documents.

4.2.2 - Operacions bàsiques

En aquest punt ariem a veure les operacions més bàsiques, per a poder treballar sobre exemples pràctics, i així disposar ja d'unes dades inicials per a practicar.

Inserció elemental

La funció inserir documents a una col·lecció. En el paràmetre posem el document directament, o una variable que contingui el document. Si la col·lecció no existeix, la creará i després alegrà el document. En la següent sentència estem treballant sobre la col·lecció `example`, que segurament ja existeix de quan van la pregunta 3.1 d'installació de MongoDB que per a proveir vam inserir un document. Però si no existeix, la crearà sense problemes.

```
> db.example.insert({ _id : "Com va la cosa?" })
{ "_id": "Com va la cosa?" }
```

Ara hem d'inserir un nou document, i així ho fa aviat ({ "message": " ", }) , xifa inserir un document). Automàticament haurà creat un element `_id` del tipus `ObjectID`, ja que li fa falta per a identificar el document entre tots els altres de la col·lecció.

I en aquest exemple ens guarden el document, i així ho fa aviat ({ "message": " ", }) , xifa inserir un document).

```
> db.example.insert({ _id : "Pesa als no ens podem quistar ..." })
{ "_id": "ObjectID('5dca31f6cd1eab1de550')", "msg": "Hola, que tal?" }
> db.example.insert(doc)
{ "_id": "ObjectID('5dca31f6cd1eab1de550')", "msg": "Pesa als no ens podem quistar ..." }
```

També ens indica que ha inserit un document. I haurà creat també el camp `_id` com veiem en el següent punt.

Lectura

Terim dues funcions per a recuperar informació: `find` i `findOne`.

- `find()`: recuperar tots els documents de la col·lecció, encara que podrem posar criteris per a que ens torni tots els documents que compleixen aquelles condicions que veuen més aviat.
- `findOne()`: només tornar un document, en principi el primer. Pot ser sobre tots els documents (per tant serà el primer document), o posar una condició, i tornarà el primer que compleix la condició.

Exemple de `findOne`:

```
> db.example.findOne()
{ "_id": "ObjectID('5dca31f6cd1eab1de550')", "msg": "Hola, que tal?" }
```

En tots els casos podem comprovar que és cert el que veníem afirmant, que ha creat automàticament l'element `_id` per a cada document (guardat). Evidentment, cadascun de nosaltres tindrà una valors diferents.

Inserció específicant el id

Ara que ja sabem consultar els documents de la col·lecció amb `find()` anem a continuar les insercions de documents, per veure les possibilitats que tenim.

En el document que hem inserit fins el moment, no hem especificat el camp `_id` i MongoDB ha generat automaticament de tipus `ObjectID`.

Però resulta que podem posar aquest camp `_id` amb el valor que vulguem. Així si, dient d'estar segurs que aquest valor no l'apareix cap altre document de la col·lecció, o ens donarà un error.

Així per exemple anem a inserir la informació d'una alumna. Es posaran en una col·lecció nova anomenada `alumnes`, i els intentarem posar un `_id` personal. Per exemple posarem els números 51, 52, 53, ...

```
> db.alumnes.insert({ _id: 51, nom: "Raquel", cognom: "Garcia Arias" })
{ "_id": "ObjectID('5dca31f6cd1eab1de550')", "nom": "Raquel", "cognom": "Garcia Arias" }
> db.alumnes.insert({ _id: 52, nom: "Raquel", cognom: "Marti Perell" })
{ "_id": "ObjectID('5dca31f6cd1eab1de550')", "nom": "Raquel", "cognom": "Marti Perell" }
> db.alumnes.insert({ _id: 53, nom: "Raquel", cognom: "Garcia Arias" })
{ "_id": "ObjectID('5dca31f6cd1eab1de550')", "nom": "Raquel", "cognom": "Garcia Arias" }

Pots veure que intentem inserir un altre document amb el mateix _id, ens donarà error:
> db.alumnes.insert({ _id: 51, nom: "Raquel", cognom: "Garcia Arias" })
{
  "error": "Error: E11000 duplicate key error collection: test.alumnes index: _id_ dup key: { _id: 51.0 }"
}
```

Ens avisa que estem duplicant el clúster principal, és a dir l'identificador.

Inserció multiple

Quan els documents que volem inserir són senzills, podem inserir més d'un a la vegada, posant els `insert()` un array amb tots els elements. En el següent exemple cream uns quants nombres primers en la col·lecció del mateix nom:

```
> db.numerosprimers.insert([ { _id:2 }, { _id:3 }, { _id:5 }, { _id:7 }, { _id:11 }, { _id:13 }, { _id:17 }, { _id:19 } ])
{ "_id": "ObjectID('5dca31f6cd1eab1de550')", "value": 2 }
{ "_id": "ObjectID('5dca31f6cd1eab1de550')", "value": 3 }
{ "_id": "ObjectID('5dca31f6cd1eab1de550')", "value": 5 }
{ "_id": "ObjectID('5dca31f6cd1eab1de550')", "value": 7 }
{ "_id": "ObjectID('5dca31f6cd1eab1de550')", "value": 11 }
{ "_id": "ObjectID('5dca31f6cd1eab1de550')", "value": 13 }
{ "_id": "ObjectID('5dca31f6cd1eab1de550')", "value": 17 }
{ "_id": "ObjectID('5dca31f6cd1eab1de550')", "value": 19 }
```

Ens avisa que ha fet les insercions, i així ho tenim:

```
> db.numerosprimers.find()
{ "_id": "ObjectID('5dca31f6cd1eab1de550')", "value": 2 }
{ "_id": "ObjectID('5dca31f6cd1eab1de550')", "value": 3 }
{ "_id": "ObjectID('5dca31f6cd1eab1de550')", "value": 5 }
{ "_id": "ObjectID('5dca31f6cd1eab1de550')", "value": 7 }
{ "_id": "ObjectID('5dca31f6cd1eab1de550')", "value": 11 }
{ "_id": "ObjectID('5dca31f6cd1eab1de550')", "value": 13 }
{ "_id": "ObjectID('5dca31f6cd1eab1de550')", "value": 17 }
{ "_id": "ObjectID('5dca31f6cd1eab1de550')", "value": 19 }
```

Eborrat

Per a eborrar un document d'una col·lecció utilitzarem la funció `remove`, passant-li com a paràmetre la condició del document o documents a eborrar:

```
> db.numerosprimers.remove({ "_id": 19 })
{ "nMatched": 1, "nRemoved": 1 }
```

Ens avisa que ha eborrat un document.

La condició no diu que siga sobre el camp `_id`. Pot ser sobre qualsevol camp, i eborrà tots els que coincideixin:

```
> db.numerosprimers.remove({ "msg": "Pesa als no ens podem quistar ..." })
{ "nMatched": 0, "nRemoved": 0 }
```

També tenim la possibilitat d'eborrar tota una col·lecció amb la funció `drop()`. Pareix atenció però que es molt sensilla d'eliminar, i per tant, potencialment molt perillosa.

```
> db.numerosprimers.drop()
{ }
```

Actualització

La funció `update` serveix per a actualitzar un document ja guardat. Tindrá dos paràmetres:

- El primer paràmetre serà la condició per trobar el document que s'ha d'actualitzar.
- El segon paràmetre serà el nou document que substituirà l'anterior.

Per exemple, si remes les dades actuals:

```
> db.example.find()
{ "_id": "ObjectID('5dca31f6cd1eab1de550')", "msg": "Hola, que tal?" }
{ "_id": "ObjectID('5dca31f6cd1eab1de550')", "msg": "Com va la cosa?" }
```

Potrem comprovar el contingut dels documents, que no li haig dit. Així anem a modificar-en el primer paràmetre posem condició de busqueda (només hi haurà un) i en el segon posem el nou document que substituirà l'anterior

```
> db.example.update({ msg: "Com va la cosa?" }, { msg: "Hola, que tal?" })
{ "nModified": 1, "nMatched": 1, "ok": 1 }
```

Observem que la contènuda del `update()` que no ha fit match (no ha pogut coincidir) amb un document, i que ha modificat un. Si en no troba cap, no donaria error, senzillament dirà que ha fit match amb 0 documents, i que ha modificat 0 documents. Mirem com efectivament ha canviat el segon document

```
> db.example.find()
{ "_id": "ObjectID('5dca31f6cd1eab1de550')", "msg": "Hola, que tal?" }
{ "_id": "ObjectID('5dca31f6cd1eab1de550')", "msg": "Què com va la cosa?" }
```

Era evident que els va canviar per les actualitzacions, ja que en moltes ocasions serà modificar freguentment el document, canviant-o alegítimament algun element. Ho podem fer comodament amb la variable: primer guardem el document a modificar en una variable, després modifiquem la variable, i per últim fem l'operació d'actualització. Evidentment si tenim alguna variable amb el contingut del document ens podrà establir el primer pas.

```
> doc = db.example.find()
{ "_id": "ObjectID('5dca31f6cd1eab1de550')", "msg": "Hola, que tal?" }
> doc.$inc = { titol: 1 }
> db.example.update({ msg: "Hola, que tal?" }, doc)
{ "nModified": 1, "nMatched": 1, "ok": 1 }

> db.example.find()
{ "_id": "ObjectID('5dca31f6cd1eab1de550')", "msg": "Hola, que tal?" }
{ "_id": "ObjectID('5dca31f6cd1eab1de550')", "msg": "titol: 1" }
```

Funció Save

Heu vist la manera d'inserir nous documents amb `insert()` i d'actualitzar documents existents amb `update()`. Però si intentem inserir un document ja existent (a manera de saber-ho per el camp `_id`) ens donarà error. I si intentem actualitzar un document no existent, no donarà error, però no actualitzarà res.

La funció `save()` us dóna la barreja dels dos: si el document que intentem a inserir ja existeix, doncs el modifiqui, i si no existeix, crea't.

Proven-ho amb un exemple, i profitarem que en la variable `doc` tenim el contingut d'un document. Fem una modificació, per exemple alegint el camp destinatari:

```
> doc = destinarari = "Ferran"
> doc
{ "destinatari": "Ferran" }
> doc.$inc = { titol: 1 }
> doc
{ "destinatari": "Ferran", "titol": 1 }
```

Ara guardem-lo amb `save()`:

```
> db.example.save(doc)
{ "nMatched": 1, "nModified": 1, "ok": 1 }
```

Ja ens avisa que com he totallat el document (`nMatched: 1`) ha modificat un. Efectivament, mimem com ha modificat el document existent:

```
> db.example.find()
{ "_id": "ObjectID('5dca31f6cd1eab1de550')", "msg": "Hola, que tal?", "titol": 1, "destinatari": "Ferran" }
{ "_id": "ObjectID('5dca31f6cd1eab1de550')", "msg": "Què com va la cosa?", "titol": 1, "destinatari": "Ferran" }
```

Així fem una modificació del camp `_id`. A tots els efectes serà un document nou, ja que la manera d'identificar un document és per aquest camp:

```
> doc._id = 100
> doc
{ "_id": 100, "msg": "Hola, que tal?", "titol": 1, "destinatari": "Ferran" }
```

I ens avem a fer el `save()` d'aquest document:

```
> db.example.save(doc)
{ "nMatched": 0, "nModified": 1, "ok": 1 }
```

Observem que ens avisa que no ha trobat cap igual, i el que fa és inserir-lo (ens diu `nMatched: 0` i ens aviat tornarem a questa paraula)

```
> db.example.find()
{ "_id": "ObjectID('5dca31f6cd1eab1de550')", "msg": "Hola, que tal?", "titol": 1, "destinatari": "Ferran" }
{ "_id": "ObjectID('5dca31f6cd1eab1de550')", "msg": "Què com va la cosa?", "titol": 1, "destinatari": "Ferran" }
```

Efectivament s'ha guardat com un nou document.

4.2.3 - Operacions d'actualització avançada

Al final de la pregunta anterior hem vist l'actualització de documents ja existents a la Base de Dades. Aquesta actualització la fem modificant tot el document, encara que tenim la variant de guardar el document en una variable, modificar aquesta variable i després fer l'actualització amb aquesta variable. Però observeu que continuem sent una modificació de tot el document, una substitució del document antic per un document nou.

En aquesta pregunta veurem la utilització d'uns modificadors (modifiers) de l'operació `update()`, que ens permeten modificar documents de forma potent: creant i eliminant claus (elements) d'un document, o canviant-los, i fins i tot afegeix o eliminar elements d'un array.

4.2.3.1 - \$set

El modificador `$set` assigna un valor a un camp del document seleccionat de la Base de Dades. Si el camp ja existia, modificalo el valor; i si no existia el creará.

La sintax del modificador `$set` és la següent:

`[$set : { [clave : valor] }]`

Però recorda que és un modificador. I hem d'utilitzar dins d'una operació d'actualització. Anirem en el segon paràmetre del `update()`, i per tant amb aquests modificadors ja posarem tot el document en el segon paràmetre, sind únicament l'operador de modificació.

Mirem-ho mitjançant un exemple:

```
[ db.alumnos.insert({ nombre:"Abel", cognome:"Benedit Carrera" })
  db.alumnos.update({ nombre:"Abel" }, { $set: { edad:21 } })
  db.alumnos.findone()
  {
    "_id": ObjectID("5dd4e30317bf4ed437db77cb"),
    "nombre": "Abel",
    "cognome": "Benedit Carrera",
    "edad": 21
  }
]
```

Suposem ara que li volem afegeir l'edat. Abans no faríem guardar el document en una variable, i afegeint el camp, per a guardar després. Ara ho tenim més fàcil:

```
[ db.alumnos.update({ nombre:"Abel" }, { $set: { edad:21 } })
  db.alumnos.findone()
  {
    "_id": ObjectID("5dd4e30317bf4ed437db77cb"),
    "nombre": "Abel",
    "cognome": "Benedit Carrera",
    "edad": 21
  }
]
```

Ha trobat un `!This` modificador. Evidentment, si hi haguera més d'un alumne a mitjans del nom Abel, els modificaria tots.

```
[ db.alumnos.findone()
  {
    "_id": ObjectID("5dd4e30317bf4ed437db77cb"),
    "nombre": "Abel",
    "cognome": "Benedit Carrera",
    "edad": 21
  }
]
```

Es pot especificar més d'un camp amb els valors correspondents. Si no existen es crearan, i si ja existen es modificaran:

```
[ db.alumnos.update({ nombre:"Abel" }, { $set: { nota: 8.5, edad:22 } })
  db.alumnos.findone()
  {
    "_id": ObjectID("5dd4e30317bf4ed437db77cb"),
    "nombre": "Abel",
    "cognome": "Benedit Carrera",
    "edad": 22,
    "nota": 8.5
  }
]
```

I res! I es pot canviar el `!spus` d'un camp determinat, i utilitzar arrays i objectes...

```
[ db.alumnos.update({ nombre:"Abel" }, { $set: { nota: [8.5,7.5,9], address:{ carrera:"Major", numero:7, op:"12001" } } })
  db.alumnos.findone()
  {
    "_id": ObjectID("5dd4e30317bf4ed437db77cb"),
    "nombre": "Abel",
    "cognome": "Benedit Carrera",
    "edad": 22,
    "nota": [
      8.5,
      7.5,
      9
    ],
    "address": [
      {
        "carrera": "Major",
        "numero": 7,
        "op": "12001"
      }
    ]
  }
]
```

Posarem fíns i tot modificar ara només el valor d'un camp d'un objecte del document. Per exemple, anirem a modificar el codi postal del anterior alumne. La manera d'ambiar al codi postal serà `adreça.op`, però haurem d'anar amb compte que viaja entre corxes per a que el trobe:

```
[ db.alumnos.update({ nombre:"Abel" }, { $set: { adreça: { op:"12001" } } })
  db.alumnos.findone()
  {
    "_id": ObjectID("5dd4e30317bf4ed437db77cb"),
    "nombre": "Abel",
    "cognome": "Benedit Carrera",
    "edad": 22,
    "nota": [
      8.5,
      7.5,
      9
    ],
    "address": [
      {
        "carrera": "Major",
        "numero": 7,
        "op": "12001"
      }
    ]
  }
]
```

El modificador **\$unset** servirà per a eliminar elements (camps) d'un o uns documents. Si el camp existeix, l'eliminarà, i si no existeix, no donarà error (avisarà que s'han modificat 0 documents).

La sintax és:

`{ $unset : {camp : 1} }`

Hauríem de posar un valor al camp que anem a esborrar per a mantenir la sintax correcta, i posarem 1 que equival a true. També podríem posar -1, que equival a false, però aleshores no l'esborraria, i per tant no faríem res. Sempre posarem 1.

Mirem un seguit exemple. Alegem un camp, que sera el numero d'ordre, i després el levarem.

```
> db.alumnes.update({nom:"Abel"}, { $set: {num_ordre:10} }, { $unset: {num_ordre: 1} })  
> db.alumnes.find()
```

```
[{"_id": "5c14741d15ddca3517b4ed437ab71d8",  
 "nom": "Abel",  
 "cognoms": "Santos Carrera",  
 "edad": 22,  
 "nota": 8.5,  
 "adres": {"  
     "correct": "Major",  
     "num": 12302},  
 "num_ordre": 10}
```

```
> db.alumnes.update({nom:"Abel"}, { $unset: {num_ordre:1} })  
> db.alumnes.find()
```

```
[{"_id": "5c14741d15ddca3517b4ed437ab71d8",  
 "nom": "Abel",  
 "cognoms": "Santos Carrera",  
 "edad": 22,  
 "nota": 8.5,  
 "adres": {"  
     "correct": "Major",  
     "num": 12302},  
 "num_ordre": 10}
```

Has d'agafar primer el camp num_ordre i hem inserit el document per comprovar que existeix. Després estarem el camp num_ordre i ens confirma que ha modificat un document. Després intentem esborrar un camp que no existeix, puntuacio. No dóna error, però ens avisa que ha modificat 0 documents. Podem comprovar al final com el document ha quedat com esperavem.

```
> db.alumnes.find()
```

```
[{"_id": "5c14741d15ddca3517b4ed437ab71d8",  
 "nom": "Abel",  
 "cognoms": "Santos Carrera",  
 "edad": 22,  
 "nota": 8.5,  
 "adres": {"  
     "correct": "Major",  
     "num": 12302},  
 "num_ordre": 10}
```

El modificador \$rename canvià el nom d'un camp. Si no existeix, no donarà error i s'encarregarà de crear-lo. Hem de posar el nou nom del camp entre comelles, per a què no doni error.

La sintax és:

```
{ $rename : { camp1 : "nou_nom1" , camp2 : "nou_nom2" , ... } }
```

Per exemple, canviem el nom del camp nota a nota:

```
> db.alineats.update( {nom:"Abel"} , { $rename : {camp1:"nota" } } )
{ "nre":1, "ok":1, "n":1, "w":1, "err":0, "modified":1 }
> db.alineats.find()
[ { "_id": ObjectID("5d6da3017bd4ed437dc77cb") ,
  "nom": "Abel",
  "carrera": "Ingenieria Carrera",
  "edad": 22,
  "nota": {
    "carrera": "MaSist",
    "nota": 9.5,
    "carrer": "MaSist",
    "carrer": "MaSist"
  },
  "telefonos": [
    9.5,
    9
  ],
  "nre": 1
} ]
```

Observiu que l'ha canviat de loc, cosa que ens fa pensar que en canviar el nom d'un camp, el que fa és tornar a crear-lo amb el nou nom, i estornar el camp antic.

En aquest exemple tornem a canviar el nom a nota, intentem canviar el nom d'un camp inexistent, camp2. No donarà error.

```
> db.alineats.update( {nom:"Abel"} , { $rename : {camp1:"nota" , camp2:"nota2" } } )
{ "nre":1, "ok":1, "n":1, "w":1, "err":0, "modified":1 }
> db.alineats.find()
[ { "_id": ObjectID("5d6da3017bd4ed437dc77cb") ,
  "nom": "Abel",
  "carrera": "Ingenieria Carrera",
  "edad": 22,
  "nota": {
    "carrera": "MaSist",
    "nota": 9.5,
    "carrer": "MaSist",
    "carrer": "MaSist"
  },
  "telefonos": [
    9.5,
    9
  ],
  "nre": 1
} ]
```

Com cabria esperar, el modificador \$inc servirà per a incrementar un camp numèric. Si el camp existeix, l'incrementarà la quantitat indicada. Si no existeix, creará el camp amb un valor inicial de 0, i incrementarà el valor amb la quantitat indicada. La quantitat pot ser positiva, negativa o fins i tot amb part fraccionària. Sempre funcionarà bé, excepte quan el camp a incrementar no siga numèric; que donarà error.

La sintax és aquella:

```
{ $inc : {camp : quantitat} }
```

En els següents exemples, incrementarem un camp nou (per tant el crearem amb el valor especificat), i després l'incrementarem en quantitats positives, negatives i fraccionàries, concientment finalitzarem amb un 2, i després incrementarem en 5, en -4 i en 2.25, per tant el resultat final serà 5.25

```
db.alumnes.update({nom : "Adel", $inc:{puntuacio:2}})
```

```
db.alumnes.update({nom : "Adel", $inc:{puntuacio:-1}})
```

```
db.alumnes.update({nom : "Adel", $inc:{puntuacio:0.5}})
```

```
db.alumnes.update({nom : "Adel", $inc:{puntuacio:2}, $inc:{notas:1}})
```

```
db.alumnes.update({nom : "Adel", $inc:{puntuacio:2}, $inc:{notas:-2}, $inc:{npq:1.125}}
```

```
db.alumnes.update({nom : "Adel", $inc:{puntuacio:2}, $inc:{notas:1}, $inc:{npq:1.125}}
```

```
db.alumnes.update({nom : "Adel", $inc:{puntuacio:5}})
```

```
db.alumnes.update({nom : "Adel", $inc:{puntuacio:-4}})
```

```
db.alumnes.update({nom : "Adel", $inc:{puntuacio:2.25}})
```

```
db.alumnes.update({nom : "Adel", $inc:{puntuacio:2}, $inc:{notas:1}, $inc:{npq:1.125}}
```

```
db.alumnes.update({nom : "Adel", $inc:{puntuacio:2}, $inc:{notas:1}, $inc:{npq:1.125}, $inc:{puntuacio:2}})
```

```
db.alumnes.update({nom : "Adel", $inc:{puntuacio:2}, $inc:{notas:1}, $inc:{npq:1.125}, $inc:{puntuacio:5}})
```

```
db.alumnes.update({nom : "Adel", $inc:{puntuacio:2}, $inc:{notas:1}, $inc:{npq:1.125}, $inc:{puntuacio:-4}})
```

```
db.alumnes.update({nom : "Adel", $inc:{puntuacio:2}, $inc:{notas:1}, $inc:{npq:1.125}, $inc:{puntuacio:2.25}})
```

```
db.alumnes.update({nom : "Adel", $inc:{puntuacio:2}, $inc:{notas:1}, $inc:{npq:1.125}, $inc:{puntuacio:2}, $inc:{notas:1}, $inc:{npq:1.125}, $inc:{puntuacio:2}})
```

```
db.alumnes.update({nom : "Adel", $inc:{puntuacio:2}, $inc:{notas:1}, $inc:{npq:1.125}, $inc:{puntuacio:2}, $inc:{notas:1}, $inc:{npq:1.125}, $inc:{puntuacio:5}})
```

```
db.alumnes.update({nom : "Adel", $inc:{puntuacio:2}, $inc:{notas:1}, $inc:{npq:1.125}, $inc:{puntuacio:2}, $inc:{notas:1}, $inc:{npq:1.125}, $inc:{puntuacio:-4}})
```

```
db.alumnes.update({nom : "Adel", $inc:{puntuacio:2}, $inc:{notas:1}, $inc:{npq:1.125}, $inc:{puntuacio:2}, $inc:{notas:1}, $inc:{npq:1.125}, $inc:{puntuacio:2.25}})
```

4.2.3.5 - Elements d'un array

Per a accedir directament a un element d'un array d'un determinat document es pot utilitzar la següent sintaxi:

```
"array_index"
```

Hem de tenir present que el primer element de l'array és el de subíndex 0. I no us oblidieu de lancer-ho tot entre comentes per a que ho puga trobar.

Si no existeix l'element amb el subíndex indicat, donarà error.

Per exemple, això així posarà la primera nota de l'alumne que estem utilitzant en tots els exemples:

```
db.alumnes.update({nom:"Abel"}, { $inc: { "nota_0": 1 } })
```

```
{ "_id": "5d6f11d78548d6fb2125e3",
  "nom": "Abel",
  "cognom": "Martí Carrera",
  "edad": 22,
  "alumne": {
    "carrera": "Matemàtiques",
    "notas": [
      {"nota": 12002}
    ],
    "notas": [
      12002,
      9500,
      9
    ]
}
```

4.2.3.6 - Inserció en Arrays: \$push

La manera més senzilla d'introduir un element en un array és utilitzar **\$push** sense més. Si existeix l'array, introduirà el o els nous elements al final. Si no existeix l'array, el creará amb aquells o aquequests elements.

La sintax és:

```
{ $push : { [clau] : [element] } }
```

Per exemple així: afegeix una nota a l'alumne de sempre, i posen-la diferent per veure que s'introdueix al final:

```
b db.alumnes.update({nom:"Abel"}, { $push : { nota : { $each:[{notat:1, opinió:"B"}, {notat:2, opinió:"A"}, {notat:3, opinió:"C"}, {notat:4, opinió:"D"}, {notat:5, opinió:"E"}, {notat:6, opinió:"F"}, {notat:7, opinió:"G"}]} } })
```

També es pot inserir un element en una determinada posició que no siga al final, però es complica prou la cosa, ja que hem d'utilitzar per una banda el modificador **\$position** per a dir on s'ha d'inserir, i per una altra banda el modificador **\$each** per a poder especificar el o els valors que es volen inserir. Es posa a continuació únicament de forma il·lustrativa.

Per a inserir en una determinada posició hem d'utilitzar obligatoriament 2 modificadors més:

- **\$position** indica a partir de quina posició es farà l'acció (normalment d'inserir en l'array, és a dir, **\$push**)
- **\$each** ens permet especificar una sèrie de valors com un array; i vol dir que es farà l'operació per a cada valor del l'array

Els dos modificadors segueixen la sintax de sempre, de clau valor, per tant el conjunt de la sintax és:

```
{ $push : { [clau]:[array],  
           "$position": [posicio],  
           "each": [valores] } }
```

Així tenim un exemple on introdueix una nota en la primera posició:

```
b db.alumnes.update({nom:"Abel"}, { $push : { nota : { $position : 0 , $each : [5] } } })
```

Nota: Si no hi ha l'array, es crearà.

4.2.3.7 - Eliminació en arrays: \$pop i \$pull

Hi ha més d'una manera d'eliminar elements d'un array.

\$pop

Si volem eliminar el primer element o l'últim, el modificador adequat és **\$pop**. La sintax és:

[**\$pop : { index : posició }**]

On en posicio podem posar:

+1 : esborrà el primer element

-1 : esborrà l'últim

En els següents exemples s'eliminarà primer l'últim element i després el primer.

```
b> db.alumnes.findOne({})  
{ "_id" : ObjectId("56d11d7d78548d0fb2f2125a3"),  
  "nom" : "Abel",  
  "cognoms" : "Serrat Carrera",  
  "adreça" : "Carrer Major",  
  "numerocell" : 7,  
  "cp" : "12002",  
  "nota" : [  
    1,  
    2,  
    3,  
    4,  
    5  
  ]  
}
```

```
b> db.alumnes.update({nom:'Abel'}, { $pop : { index: 1 } })  
WriteResult: { nModified: 1, upserted: { _id: '56d11d7d78548d0fb2f2125a3' }}
```

```
b> db.alumnes.update({nom:'Abel'}, { $pop : { index: -1 } })  
WriteResult: { nModified: 1, upserted: { _id: '56d11d7d78548d0fb2f2125a3' }}
```

```
b> db.alumnes.update({nom:'Abel'}, { $pop : { index: 1 }, $pop : { index: -1 } })  
WriteResult: { nModified: 2, upserted: { _id: '56d11d7d78548d0fb2f2125a3' }}
```

```
b> db.alumnes.update({nom:'Abel'}, { $pop : { index: 1 }, $pop : { index: -1 } })  
WriteResult: { nModified: 2, upserted: { _id: '56d11d7d78548d0fb2f2125a3' }}
```

\$pull

Amb aquest modificador esborrà els elements de l'array que coincideixen amb una condició, estiguin en la posició que estiguin. Observeu com es pot eliminar més d'un element.

Per a poder comprovar-lo bé, primer inserim un altre element al final de l'array, amb el valor 7.5 (n'heu seguit els mateixos exemples que en aquells apunts, aquest valor ja es troba en la segona posició).

```
b> db.alumnes.update({nom:'Abel'}, { $push : { nota : 7.5 } })  
WriteResult: { nModified: 1, upserted: { _id: '56d11d7d78548d0fb2f2125a3' }}
```

```
b> db.alumnes.findOne({})  
{ "_id" : ObjectId("56d11d7d78548d0fb2f2125a3"),  
  "nom" : "Abel",  
  "cognoms" : "Serrat Carrera",  
  "adreça" : "Carrer Major",  
  "numerocell" : 7,  
  "cp" : "12002",  
  "nota" : [  
    1,  
    2,  
    3,  
    4,  
    5,  
    7.5  
  ]  
}
```

Ara anem a esborrar amb **\$pull** l'element de valor 7.5

```
b> db.alumnes.update({nom:'Abel'}, { $pull : { nota : 7.5 } })  
WriteResult: { nModified: 1, upserted: { _id: '56d11d7d78548d0fb2f2125a3' }}
```

```
b> db.alumnes.findOne({})  
{ "_id" : ObjectId("56d11d7d78548d0fb2f2125a3"),  
  "nom" : "Abel",  
  "cognoms" : "Serrat Carrera",  
  "adreça" : "Carrer Major",  
  "numerocell" : 7,  
  "cp" : "12002",  
  "nota" : [  
    1,  
    2,  
    3,  
    4,  
    5  
  ]  
}
```

Aquesta paraula ja l'hem començat en un punt anterior.

En el `update()` normal, si la condició de búsquedera no donava cap resultat (parlant d'índex, si no feia matching amb cap document), doncs no actualitzava cap document i punt.

El `Upsert` és una variant de l'update, que quan no coincideix cap document amb la condició, crearà un document nou que serà el resultat de combinar el criteri que s'ha utilitzat en la condició amb les operacions d'actualització fets en el segon paràmetre.

Per a que un `Update` actue d'aquesta manera, li hem de posar un tercer paràmetre amb el valor `true`:

```
update( { ... } , { ... } , true )
```

Recorda que el primer paràmetre era la condició, i el segon l'actualització.

Mirem-ho en l'exemple dels alumnes. Si avenim a actualitzar els cognoms, i es troba el document, s'actualitzarà:

```
db.alumnes.update( { nom: "Abei" } , { $set: { cognom: "Bennat Cantera" } } , true )  
WriteResult({ "nModified": 0, "nUpserted": 1, "nDeleted": 0 })
```

Efectivament, ens diu que ha modificat un document.

Pero si no es troba el document (per exemple perquè li hem posat el nom `Berta`):

```
db.alumnes.update( { nom: "Berta" } , { $set: { cognom: "Bennat Cantera" } } , true )  
WriteResult({ "nModified": 0, "nUpserted": 1, "nDeleted": 0 })  
{ "_id": ObjectId("56d4dbd136db9d9cb4bd51a") }
```

Já ens avisa que no ha fet cap matching, i ha fet un `Upsert`. Ho podem comprovar mirant tots els documents de la col·lecció:

```
db.alumnes.find()  
[ { "_id": ObjectId("56d4dbd136db9d9cb4bd51a"), "nom": "Abei", "cognom": "Bennat Cantera", "adat": 22, "adreca": { "carrer": "Major", "numero": 7, "cp": "12550" }, "nota": { 9.5, 9 } } ]
```

Nou document vindrà els camps:

- `_id`, amb el que ens havia avisat que generaria
- Els camps de la condició, que en el nostre exemple de `{ nom: "Berta" }`
- Els camps de l'actualització, que en el nostre exemple arren els cognoms

En la pregunta anterior hem vist com introduir, eliminar i modificar documents. Les consultes de documents han seguit molt senzilles, per a comprovar únicament els resultats. En aquesta pregunta veurem en profunditat la consulta de documents.

- Funcions `$map` i `$findDepth`, que són les que hem utilitzat fins ara. Veurem en profunditat la seua sintaxi i potència.
- L'ús d'unes i altres queries tancades als resultats.
- Fins i tot podrem elaborar més els resultats, agrupant els resultats, utilitzant funcions d'agregació (o més dir operadors d'agregació) i donant-los un aspecte diferent.

4.3.1 - Paràmetres de les funcions find() i findOne()

Les funcions `find()` i `findOne()` són absolutament equivalents, amb l'única diferència que la primera torna tots els documents trobats, mentre que la segona només torna el primer document trobat.

Per una millor comprensió, utilitzarem únicament `findOne()`, per veure tots els resultats obtinguts.

La funció `findOne()` s'ha comparat tradicionalment amb la sentència `SELECT` de SQL. Sempre tornarà un conjunt de documents, que poden variar des de no tornar cap document, a tornar-los tots els de la col·lecció.

La funció `findOne()` pot tenir uns quants paràmetres:

- El primer indica una condició o criteri, i tornarà aquells documents de la col·lecció que compleixen la condició o criteri. Aquesta condició es donada en forma de document (o objecte) JSON, i és com havíem vist en la funció `update()`.

```
[db].collection("clients").findOne({ "client": "valori1" })
```

Tornarà tots els documents de la col·lecció `clients` que tinguin el camp `client` i que en ella tinguen el valor `valori1`. Aquest criteri pot ser el complicat que faca falta, formant-lo en JSON. En definitiva, tornarà aquells documents que facen matching amb el document del criteri, és a dir, funcionaria com un `AND`.

```
[db].collection("clients").findOne({ "client": "valori1", "client": "valori2" })
```

que tornaria aquells documents de la col·lecció que tinguin el camp `client` amb el valor `valori1` i que tinguin el camp `client` amb el valor `valori2`.

Si no volem posar cap criteri, per a que els torna tots, no posarem res com a paràmetre, o encara millor, i passarem un document (objecte) buit, de manera que tots els documents de la col·lecció faran matching amb ell.

```
[db].collection("clients").findOne({ })
```

Tindrem això present, sobretot quan ens hoquem utilitzar el segon paràmetre de `findOne()`. Si no volem cap criteri, posarem el document buit com l'exemple anterior.

- El segon paràmetre ens servirà per a definir els camps dels documents que es tornaran. També linda al format JSON d'un objecte al qual li posarem com a claus els diferents camps que volem que apareguen o no, i com a valor 1 per a que si que apareguen i 0 per a que no apareguen.

Si posem algun camp a que si que aparegui (ve a dir, amb el valor 1), els únics que apareixeran seran aquests, a més del `_id` que per defecte sempre apareix.

Per tant si no volem que aparegui `_id` posarem:

```
[db].collection("clients").findOne({ "client": "valori1" }, { "client": 0, "name": 1, "age": 1, "address": 1, "phone": 1, "notes": 1 })
```

I si volen traure únicament el nom:

```
[db].collection("clients").findOne({ "client": "valori1" }, { "name": 1 })
```

```
[db].collection("clients").findOne({ "client": "valori1" }, { "name": 1, "age": 0 })
```

Per últim, com que això d'arriba utilitzarem documents més complicats, si volem que ens apareguen els camps que retornem d'una forma un poc més elegant o bonica (pretty), posarem aquesta funció al final: `findOne("pretty")`

```
[db].collection("clients").findOne({ "client": "valori1" }, { "name": 1, "age": 1, "address": 1, "phone": 1, "notes": 1 }).pretty()
```

```
[db].collection("clients").findOne({ "client": "valori1" }, { "name": 1, "age": 1, "address": 1, "phone": 1, "notes": 1 }).pretty()
```


| "id": "#978400342155", "titulo": "El juego de pipper", "editorial": "Plaza & Janés", "precio": "21.75"}
| "id": "#978400200815", "titulo": "Juego de tronos. Canción de hielo y Fuego I", "editorial": "Gigamesh", "precio": "9.5"}
Si el numero que se consulta es menor que el que se han leido, se muestra el anterior. Por ejemplo, de la editorial Planeta nos ha ido los libros. Encara que poseen `lim(3)`, se han tomado 2.

| > libro.find("editorial", "Planeta")
[{"id": "978400342155", "titulo": "El juego de pipper", "editorial": "Plaza & Janés", "precio": "21.75"}]
| > libro.find("editorial", "Planeta")
[{"id": "978400342155", "titulo": "El juego de pipper", "editorial": "Plaza & Janés", "precio": "21.75"}]

skipped
Se saltaran los primeros `n` documentos. Si hi ha alguno més documents dels que se salten, donc no se'n mostraran cap.

| > libro.find("editorial", "Planeta").skip(1).skip(1).skip(1)
[]
| > libro.find("editorial", "Planeta").skip(1).skip(1).skip(1).skip(1)
[{"id": "978400342155", "titulo": "El juego de pipper", "editorial": "Plaza & Janés", "precio": "21.75"}]
| > libro.find("editorial", "Planeta").skip(1).skip(1).skip(1).skip(1).skip(1)
[{"id": "978400342155", "titulo": "El juego de pipper", "editorial": "Plaza & Janés", "precio": "21.75"}]
| > libro.find("editorial", "Planeta").skip(1).skip(1).skip(1).skip(1).skip(1).skip(1)
[{"id": "978400342155", "titulo": "El juego de pipper", "editorial": "Plaza & Janés", "precio": "21.75"}]
| > libro.find("editorial", "Planeta").skip(1).skip(1).skip(1).skip(1).skip(1).skip(1).skip(1)
[{"id": "978400342155", "titulo": "El juego de pipper", "editorial": "Plaza & Janés", "precio": "21.75"}]

```

sortid
Serve per a ordenar. Com a paràmetre se li passarà un objecte JSON amb les claus per a ordenar, i els valors seran:
  * 1: ordre ascendent
  * -1: ordre descendent

Si posem més d'una clau, l'ordenarà pel primer, en cas d'empat pel segon, ...

En aquest exemple ordenem per preu

GET /llibre?find[{:}, {:title|1}, {:preu|1}, :editorial|1]]).sort[{:preu:-1}]

  * "id" : 97984462001915, "title" : "Jugando con el diablo", "editorial" : "Obedientia", "preu": 9.45
  * "id" : 97984462001915, "title" : "Juega de tronos: Canción de hielo y Fuego 1", "editorial" : "Gigamesh", "preu": 9.95
  * "id" : 97984462001915, "title" : "El Principito", "editorial" : "Obedientia", "preu": 9.45
  * "id" : 97984462001915, "title" : "Juego de tronos", "editorial" : "Gigamesh", "preu": 9.95
  * "id" : 97984462001915, "title" : "Las Reglas del Juego", "editorial" : 15.95
  * "id" : 97984462001915, "title" : "El Principito", "editorial" : "Obedientia", "preu": 17.23
  * "id" : 97984462001915, "title" : "Cien Mañanas", "editorial" : "Planeta", "preu": 21.75
  * "id" : 97984462001915, "title" : "El Principito", "editorial" : "Planeta", "preu": 21.75

Com deseu, es pot ordenar per editorial en ordre ascendent: per preu en ordre descendent:
GET /llibre?find[{:}, {:title|1}, {:preu|1}, :editorial|1]]).sort[{:editorial|1}]


```

L'agregació ens permetrà fer consultes molt avançades. Es un procés un poc complicat però molt potent. Ens donarà una potència quasi com la del SQL quan comencem a utilitzar el GROUP-BY i HAVING.

La tècnica que s'utilitza és del pipeline, és a dir un sèrie de comandes, cadauna agafa els dades que proporciona l'anterior i la seua regada proporciona les dades del següent comando. D'aquesta manera es tractarà un conjunt de documents i es faran "operacions" sobre ell seqüencialment en técs: filtrat, projectat, agrupacions, ordenació, limitació i skipping (saltar algunes).

La sintax serà:

```
[db].[col].select.aggregate([operator_b1, operator_b2, operator_b3, ..., operator_bn], [operator_Smatch], [operator_Sskip], [operator_Slimit])
```

L'ordre dels operadors pot canviar, però hem de tenir en compte que els comandes s'executarán en el ordre en el qual els posem (desqueva a dreta). Així, per exemple, pot ser molt convenient posar el primer operador el Smatch, que és el de seleccionar documents, així les altres operacions es faran sobre menys documents i aniran més ràpids.

Cada parametre del aggregate, és a dir, cada operador troba format JSON. I per tant sempre serà de l'estil:

```
[{"operator": "[col].value", ... }]
```

Operador \$match

Serveix per a filtrar els documents. Aleshores, l'agregació només afectarà als documents seleccionats. Es poden utilitzar tots els operadors que hem anat estudiant.

El següent exemple selecciona els documents de l'editorial Planeta. Ho fa per mitjà de aggregate, però com no hi ha res més, senzillament selecciona els documents.

```
[db].[libro].aggregate([{$match:{editorial:'Planeta'}}])
```

En el següent exemple, a més de seleccionar els de l'editorial Planeta després apliquem una projecció sobre els camps titol i editorial, per a poder visualitzar millor el resultat.

```
[db].[libro].aggregate([{$match:{editorial:'Planeta'}}, {$project:{titol:1, editorial:1}}, {"$limit":100}])
```

Operador \$project

Es permet projectar sobre determinats camps del document, però és molt més compacte que en el projecto "normal" que havíem fet fins ara, ja que permet també renomencar camps, fer càlculs, etc.

Project

La manera més senzilla és projectar sobre algun camp dels existents, i el funcionament és idèntic al de l'altra vegada (valors 1 per a que apareguen, 0 per a que no apareguen; per defecte, _id sempre apareix)

```
[db].[libro].aggregate([{$project:{titol:1, editorial:1}}, {"$limit":100}])
```

Renomenc

S'projeta també ens permet renomencar camps existents (després veurem que també càlculs). La manera serà posar d'aquesta manera:

```
[{"$project": [{"name": "new_name": {"$eq": "old_val1"}, "name": "new_val1"}]}
```

El select està en el darrer que va davant del camp vell, ja que d'aquesta manera ens reforma el valor d'aquest camp. Així per exemple renomencem el camp estatock a disponible, a banda de traure el títol

```
[db].[libro].aggregate([{$project:{titol:1, disponibile:'$estatock'}}, {"$limit":100}])
```

Calculats

Amb aquest nom generic ens referim a tots els càlculs, expressions i moltes cases que podem posar per a transformar el que ja tenim. Com veiem, aqüí és molt més potent que el projecto normal.

Expressions matemàtiques

Per exemple, traurem títol del llibre, preu i preus en milions (multiplicant per 1000000).

```
[db].[libro].aggregate([{$project:{titol:1, preu:'$preu * 1000000', preus_milions:'$preus * 1000000'}}, {"$limit":100}])
```

Expressions de strings

Ens permeten manipular els strings per a extreure subcadenes, concatenar, passar a majúscules o minúscules. Aquestes són algunes de les funcions:

```
* $Substr : ($exp, $inc, $length) extraiu una subcadena del string del primer parametre, de la posició que indica el segon parametre (primer caràcter) i l'ants caràcters com el tretar parametre
```

* \$concat : (\$exp1, \$exp2,...) concatena les expressions que hi ha en l'any

* \$Replace : (\$exp, \$oldValue, \$newValue) substitueix els valors dels quals volen agrupar

Per exemple, anem a traure el títol dels llibres amb l'autor entre parèntesis

```
[{"$libros": [{"titol": "$titol", "autor": "($titol)", "libre": "($titol)"}, {"$libros": [{"titol": "Círculo Maximo (Santiago Posteguillo)", "autor": "Círculo Maximo (Santiago Posteguillo)"}, {"$libros": [{"titol": "Juego de tronos: Canción de hielo y Fuego I (George R.R. Martin)", "autor": "Juego de tronos: Canción de hielo y Fuego I (George R.R. Martin)"}, {"$libros": [{"titol": "Juego de tronos: Canción de hielo y Fuego II (George R.R. Martin)", "autor": "Juego de tronos: Canción de hielo y Fuego II (George R.R. Martin)"}, {"$libros": [{"titol": "La princesa de hielo (Camilla Läckberg)", "autor": "La princesa de hielo (Camilla Läckberg)"}, {"$libros": [{"titol": "La princesa de hielo (Camilla Läckberg)", "autor": "La princesa de hielo (Camilla Läckberg)"}, {"$libros": [{"titol": "Las reglas del juego (Anna Casanova)", "autor": "Las reglas del juego (Anna Casanova)"}
```

Operador \$group

Realitzar grups sobre els documents seleccionats prèviament, per a valors iguals del camp o expressions que determinen. Posteriorment, amb els grups, podem realitzar operacions, com sumar o traure la mitjana d'alguna quantitat dels documents del grup, o el màxim o mínim,...

Per a poder agrupar, hem de definir com a _id del grup el camp o camps pel valor dels quals volen agrupar. Per exemple, si volem agrupar els llibres per l'editorial, hauríem de definir el _id del grup el camp editorial.

```
[$group: {_id: {camp: "campo_a_agrupar"}, ...}]
```

S'agrupen per un únic camp, senzillament el posen amb un valor dolent i entre cometes. Si és més d'un camp, els posen com un objecte. Per exemple, agrupem per editorial:

```
[db].[libro].aggregate([{$group: {_id: {"camp": "Editorial"}, "any": 1}}, {"$group: {_id: {"camp": "Editorial"}, "any": 1, "year": 2009}}, {"$group: {_id: {"camp": "Editorial"}, "any": 1, "year": 2010}}, {"$group: {_id: {"camp": "Editorial"}, "any": 1, "year": 2011}}, {"$group: {_id: {"camp": "Editorial"}, "any": 1, "year": 2012}}, {"$group: {_id: {"camp": "Editorial"}, "any": 1, "year": 2013}}, {"$group: {_id: {"camp": "Editorial"}, "any": 1, "year": 2014}}])
```

I ara agrupem per editorial i any de publicació (els dos llibres de Planeta són del 2013)

```
[db].[libro].aggregate([{$group: {_id: {"camp": "Editorial"}, "any": 1, "year": 2013}}, {"$group: {_id: {"camp": "Editorial"}, "any": 1, "year": 2014}}, {"$group: {_id: {"camp": "Editorial"}, "any": 1, "year": 2015}}, {"$group: {_id: {"camp": "Editorial"}, "any": 1, "year": 2016}}, {"$group: {_id: {"camp": "Editorial"}, "any": 1, "year": 2017}}, {"$group: {_id: {"camp": "Editorial"}, "any": 1, "year": 2018}}, {"$group: {_id: {"camp": "Editorial"}, "any": 1, "year": 2019}}])
```

Operador \$grouped

Els permeten fer alguna operació sobre els documents del grup. Es posen com a segon parametre del grup (després de la definició del _id).

- * \$sum : valor : sumaria el valor de tots els documents del grup. El valor pot ser un camp numèric, o alguna altra cosa més complicada.

- * \$avg : calcula la mitjana dels valors per als documents del grup

- * \$max : maximum

- * \$min : minimum

- * \$first : agrupa el primer valor de l'expressió del grup, ignorant les altres del grup

- * \$last : agrupa l'últim

Per exemple, la suma dels preus dels llibres de cada editorial:

```
[db].[libro].aggregate([{$group: {_id: {"camp": "Editorial"}, "any": 1, "year": 2013}}, {"$group: {_id: {"camp": "Editorial"}, "any": 1, "year": 2014}}, {"$group: {_id: {"camp": "Editorial"}, "any": 1, "year": 2015}}, {"$group: {_id: {"camp": "Editorial"}, "any": 1, "year": 2016}}, {"$group: {_id: {"camp": "Editorial"}, "any": 1, "year": 2017}}, {"$group: {_id: {"camp": "Editorial"}, "any": 1, "year": 2018}}, {"$group: {_id: {"camp": "Editorial"}, "any": 1, "year": 2019}}])
```

Si observem com hi així podem fer que no té editorial.

Així agrupem per any de publicació l'atribut que fecha:

```
[db].[libro].aggregate([{$group: {_id: {"camp": "any"}, "fecha": 1}}])
```

Si agrupem per un únic camp, senzillament el posen amb un valor dolent i entre cometes. Si és més d'un camp, els posen com un objecte. Per exemple, agrupem per editorial:

```
[db].[libro].aggregate([{$group: {_id: {"camp": "Editorial"}, "any": 1}}, {"$group: {_id: {"camp": "Editorial"}, "any": 1, "fecha": 1}}, {"$group: {_id: {"camp": "Editorial"}, "any": 1, "fecha": 2009}}, {"$group: {_id: {"camp": "Editorial"}, "any": 1, "fecha": 2010}}, {"$group: {_id: {"camp": "Editorial"}, "any": 1, "fecha": 2011}}, {"$group: {_id: {"camp": "Editorial"}, "any": 1, "fecha": 2012}}, {"$group: {_id: {"camp": "Editorial"}, "any": 1, "fecha": 2013}}, {"$group: {_id: {"camp": "Editorial"}, "any": 1, "fecha": 2014}}])
```

I ara agrupem per editorial i any de publicació (els dos llibres de Planeta són del 2013)

```
[db].[libro].aggregate([{$group: {_id: {"camp": "Editorial"}, "any": 1, "year": 2013}}, {"$group: {_id: {"camp": "Editorial"}, "any": 1, "year": 2014}}, {"$group: {_id: {"camp": "Editorial"}, "any": 1, "year": 2015}}, {"$group: {_id: {"camp": "Editorial"}, "any": 1, "year": 2016}}, {"$group: {_id: {"camp": "Editorial"}, "any": 1, "year": 2017}}, {"$group: {_id: {"camp": "Editorial"}, "any": 1, "year": 2018}}, {"$group: {_id: {"camp": "Editorial"}, "any": 1, "year": 2019}}])
```

Operador \$grouped

Serveix per a ordenar i seguir la mateixa similitud que en les consultes normal (1: ordre ascendent, -1: ordre descendent). Podrem ordenar pels camps normals o per camps calculats.

Per exemple ordenem per la suma de preus de cada editorial:

```
[db].[libro].aggregate([{$group: {_id: {"camp": "Editorial"}, "any": 1, "year": 2013}}, {"$group: {_id: {"camp": "Editorial"}, "any": 1, "year": 2014}}, {"$group: {_id: {"camp": "Editorial"}, "any": 1, "year": 2015}}, {"$group: {_id: {"camp": "Editorial"}, "any": 1, "year": 2016}}, {"$group: {_id: {"camp": "Editorial"}, "any": 1, "year": 2017}}, {"$group: {_id: {"camp": "Editorial"}, "any": 1, "year": 2018}}, {"$group: {_id: {"camp": "Editorial"}, "any": 1, "year": 2019}}])
```

I els ordenei en forma descendente per la mitjana de preus de cada any.

```
[db].[libro].aggregate([{$group: {_id: {"camp": "any"}, "year": 1, "fecha": 1}}, {"$group: {_id: {"camp": "any"}, "year": 1, "fecha": 2009}}, {"$group: {_id: {"camp": "any"}, "year": 1, "fecha": 2010}}, {"$group: {_id: {"camp": "any"}, "year": 1, "fecha": 2011}}, {"$group: {_id: {"camp": "any"}, "year": 1, "fecha": 2012}}, {"$group: {_id: {"camp": "any"}, "year": 1, "fecha": 2013}}, {"$group: {_id: {"camp": "any"}, "year": 1, "fecha": 2014}}])
```

Operador \$skip

Limita el resultat del agregació al numero indicat.

Per exemple, els tres anys de mitjana de preus més cara. És com l'últim exemple, alegant el limit:

```
[db].[libro].aggregate([{$group: {_id: {"camp": "any"}, "year": 1, "fecha": 1}}, {"$group: {_id: {"camp": "any"}, "year": 1, "fecha": 2009}}, {"$group: {_id: {"camp": "any"}, "year": 1, "fecha": 2010}}, {"$group: {_id: {"camp": "any"}, "year": 1, "fecha": 2011}}, {"$group: {_id: {"camp": "any"}, "year": 1, "fecha": 2012}}, {"$group: {_id: {"camp": "any"}, "year": 1, "fecha": 2013}}, {"$group: {_id: {"camp": "any"}, "year": 1, "fecha": 2014}}], {"$skip": 3})
```

Operador \$limit

Serveix per a limitar el resultat.

En l'exemple anterior, ara s'afegeix els primers 3 primers:

```
[db].[libro].aggregate([{$group: {_id: {"camp": "any"}, "year": 1, "fecha": 1}}, {"$group: {_id: {"camp": "any"}, "year": 1, "fecha": 2009}}, {"$group: {_id: {"camp": "any"}, "year": 1, "fecha": 2010}}, {"$group: {_id: {"camp": "any"}, "year": 1, "fecha": 2011}}, {"$group: {_id: {"camp": "any"}, "year": 1, "fecha": 2012}}, {"$group: {_id: {"camp": "any"}, "year": 1, "fecha": 2013}}, {"$group: {_id: {"camp": "any"}, "year": 1, "fecha": 2014}}], {"$limit": 3})
```


Exercici 7.1 (Firebase)

Farem una aplicació per a crear un **chat** entre el grup de classe. Per a poder compartir els dades entre tots, accedirem a una altra Base de Dades d'un altre usuari, tots utilitzarem el mateix:

• Compte de Google: lesejcamines.ad@gmail.com

• Contraseña: ad_lesejcamines

En el moment només tenim la Base de Dades **chat-ad**, la referència de la qual és <https://chat-ad.firebaseio.com/>

PODEU PER L'APLICACIÓ EN ECLIPSE O EN ANDROID

Aplicació en Eclipse

Podeu utilitzar l'**'Esquella'** que hi ha a continuació. En ell teniu:

- Una etiqueta per a indicar nom d'usuari.
- Un **'EditText'** per a introduir el nom d'usuari. Aquestes dues coses han d'anar dalt de tot, en la mateixa línia.
- Un altre **'EditText'** per a introduir el chat.
- Un altre **'EditText'** per a introduir el nou text de conversa.
- Un botó d'**'Enviar'**. Aquelles dues coses han d'anar de baix de tot, en la mateixa línia.

Hanurem d'incorporar el botó junt amb tota la configuració i la clau privada de la connexió. El fitxer s'anomena **chat-ad.firebaseio-adminsdk-my2d-8cd9944034.json**, i el tenim com un recurs en fàlvia virtual.

L'aplicació ha de guardar el missatge i moment d'apartar el botó en la llista **chat**. Un missatge consta del nom de la persona que l'escriu (l'afegem del **EditText** de dalt), el moment en què es fa (l'afegeiem de la data del sistema) i el contingut del missatge. Per a que siguin compatibles totes les aplicacions, han de tenir el mateix format, que serà:

• **nom (String)**: conté el nom de la persona.

• **date (long)**: conté la data-hora del missatge.

• **contingut (String)**: el text del missatge.

S'han de mostrar tots els elements de la llista **chat**, que seran tots els missatges i actualitzat al moment en què es crea un nou missatge (un nou element de la llista) amb aquest format:

User (data): message

A continuació teniu l'esquella del programa que podeu utilitzar:

```
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Container;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.IOException;
import java.io.InputStream;
import java.text.SimpleDateFormat;
import java.util.Date;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.JTextField;
import javax.swing.WindowConstants;

import com.google.api.auth.oauth2.GoogleCredentials;
import com.google.firebase.FirebaseApp;
import com.google.firebase.FirebaseOptions;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.DatabaseSnapshot;
import com.google.firebase.database.DatabaseError;
import com.google.firebase.database.FirebaseDatabase;
import com.google.firebase.database.ValueEventListener;

public class Pantalla_Exercici7_1 extends JFrame implements ActionListener {
    private static final long serialVersionUID = 1L;
    JLabel et_usu = new JLabel("Nom d'usuari:");
    JTextField usuari = new JTextField(15);
    JTextField message = new JTextField(15);
    JButton enviar = new JButton("Enviar");

    // en iniciar posen un contendor per als elements anteriors
    public void init() throws IOException {
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setLayout(new BorderLayout());
        // contendor per als elements
        JPanel panel1 = new JPanel(new GridLayout(1, 1));
        panel1.setLayout(new BorderLayout());
        panel1.add(usuari);
        getContentPane().add(panel1, BorderLayout.NORTH);

        JPanel panel2 = new JPanel(new BorderLayout());
        panel2.setLayout(new BorderLayout());
        area.setForeground(Color.blue);
        jScrollPane2 = new JScrollPane(area);
        jScrollPane2.setVerticalScrollBarPolicy(ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS);
        jScrollPane2.setHorizontalScrollBarPolicy(ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER);
        getContentPane().add(jScrollPane2, BorderLayout.CENTER);

        JPanel panel3 = new JPanel(new BorderLayout());
        panel3.setLayout(new BorderLayout());
        panel3.add(message);
        panel3.add(enviar);
        getContentPane().add(panel3, BorderLayout.SOUTH);

        setVisible(true);
        enviar.addActionListener(this);
    }

    FileInputStream serviceAccount = new FileInputStream(
            "chat-ad.firebaseio-adminsdk-my2d-8cd9944034.json");

    FirebaseOptions option = new FirebaseOptions.Builder()
            .setCredentials(GoogleCredentials.fromStream(serviceAccount))
            .setDatabaseUrl("https://chat-ad.firebaseio.com").build();

    FirebaseApp.initializeApp(option);

    DatabaseReference refat = FirebaseDatabase.getInstance().getReference("chat");

    refat.addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(DataSnapshot dataSnapshot, String s) {
            // S'entrenen per a carregar les dades del chat al JTextArea area
        }

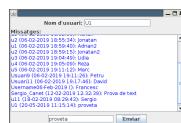
        @Override
        public void onCancelled(DatabaseError databaseError) {
        }
    });
}

@Override
public void actionPerformed(ActionEvent e) {
    // S'entrena per a guardar el missatge
}
```

I així el programa principal:

```
import java.io.IOException;
public class Exercici7_1 {
    public static void main(String[] args) throws IOException {
        final Pantalla_Exercici7_1 finestra = new Pantalla_Exercici7_1();
        finestra.init();
    }
}
```

I aquest seria el resultat:



Bases de Dades

<http://localhost:51235/temp> print dirs/eXeTemp...

```
    int[] solucio = new int[2];
    solucio[0]=pos;
    solucio[1]=nupos;
    return solucio;
}
```

 Exercici 7.3 (Redis)

En un nou paquet anomenat **Exercici** dins del projecte **Tema7_1**, crea la classe **ConsultarClaus.java**, amb **main**, que permeta consultar totes les claus guardades en el nostre servidor Redis.

Ha de presentar totes les claus actuals en Redish i el seu tipus, però amb un número davant. Posteriorment ha de demanar un número per teclat, i presentar la clau i el valor corresponent al número introduït, fins introduir el 0. Observa que depenent del tipus de la clau s'haurà de fer d'una manera o una altra. En la imatge teniu un exemple de cada

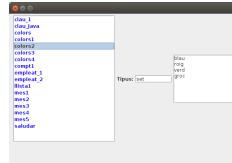
 Exercici 7.4 (Redis)

Realitzar en el mateix paquet un altre programa, aquesta vegada gràfic, que ens diga el mateix, però d'una forma més atractiva.

- El programa principal s'ha de de **ConsultaClausOralia**
 - On entrarà al programa s'ha de de **ConsultaClausOralia_Pantalla**.
 - Confirmando un **JList** han d'apareixer totes les claus, millor ordenades alfabèticament. Recordeu que el **JList** és un poc complicat, que es basa en un **DefaultListModel**. I és aquest a qui haudeu d'aigregar els elements.
 - Al costat d'haver un **JTextFiel** que diga de quin tipus són quan se selecciona un element del **JList**

- I també un JTextArea

I aquest el seu aspecte



Exercici 7.5 (Redis) (voluntari)

Posteriorment inserim el nom i l'edat amb la meua en un cert nombre i l'estadística (Resultat) anomenar **les_marmes**, on utilitzarem nom va al valor del binàriar. Com a resultat d'això hem de veure que els valors de **les_marmes** es renoven cada cop que fem una nova introducció d'una edat.

- Posteriorment prenguis el nom i guardis el en la marca d'un temps breu (`setit` serà el interval que serveix per a durar, en temps. Com a complement inserim que en un temps de `setit` s'ha de fer una pausa de `pausa` segons).

Exercici 7.6 (MongoDB)

Anula l'existència del Principi de la causalitat adhesiva

10.2.3. Membres

<http://www.technet.microsoft.com>

En l'annex del WebService tens la manera d'accedir a un WebService.

• Es un programa anomenat **MongoDB** (amb main) que introdueix en **MongoDB** cada estació com un document en una col·lecció anomenada **bicicletas**.
• Es un altre programa anomenat **Montreries** que agafa tots els documents de la col·lecció **bicicletas** de la Base de Dades de **MongoDB**, que són totes les estacions, i trageix la seua informació amb aquest aspecte (només es mostren les 10 primeres).

Exercici 7.7 (Firebase) Voluntari

Per el joc del 3 en ratlla (TicTacToe) fet en PMDM, però per a 2 jugadors que juguen en terminals diferents. El primer que hauran de fer és connectar-se, i després començar a jugar. Ens guardarem en Firebase (dins de l'objecte TicTacToe) tota la informació necessària.

- En qualsevol l'estat de la partida en **StateGame**, que podrà tenir els valors Playing, Draw o Win, s'informarà al usuari que està en **Connecting**. Inicialment potencia considerar que l'estat de Draw es quedarà a la informació de 2 jugadors (més) i intentarà trobar un guanyador.
 - Quan el primer jugador es connecta a l'enemic o jugador #2 (per exemple amb els valors 2 0), el valor **value** canvia a **Connecting**. Quan es connecta el segon s'inicialitzarà el jugador #2 (per exemple amb els valors 2 0). L'estat passarà a ser **Playing**.
 - Si el jugador 1 quede sol (el seu contrincant no hi està), es tornarà a **Connecting**.
 - Si el jugador 1 gana, obtindrà la victòria, si no, la derrota. Si el mateix es fa amb el jugador 2, el resultat serà el contrari. Si es torna a **Connecting**, el resultat serà el mateix.
 - Quan finalitza la partida, a banda de marcar el guanyador que no hi ha, se'n informarà el resultat de la partida (win o loss) i el resultat de la nostra part (win o loss).