Thực hành tuần 10

DataStore

implementation "androidx.datastore:datastore-preferences:1.0.0"

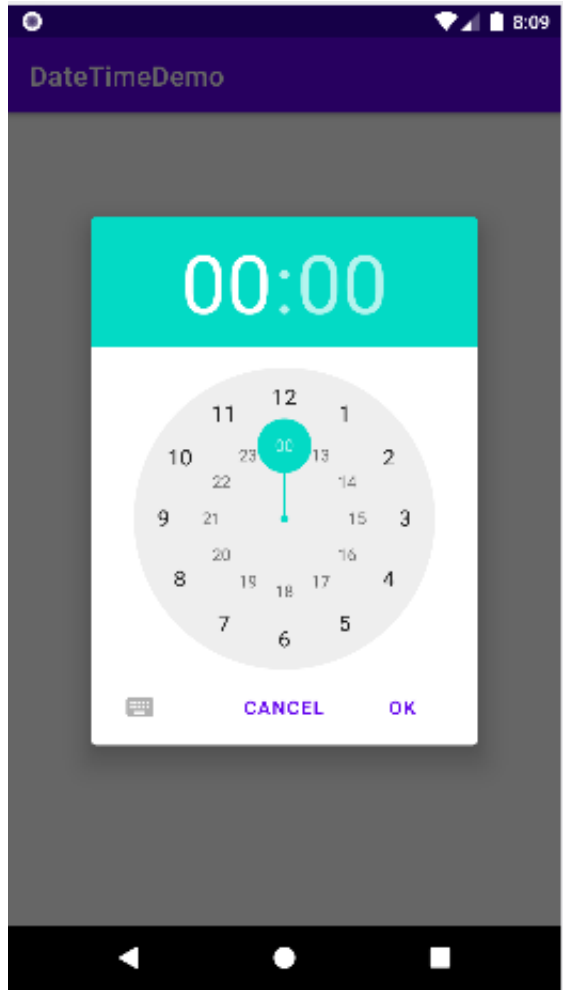
```
val Context.dataStore: DataStore<Preferences> by
preferencesDataStore(name = "settings")
class MainActivity : AppCompatActivity() {
    lateinit var binding: ActivityMainBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)
        binding.btnSave.setOnClickListener {
            CoroutineScope(Dispatchers.IO).launch {
                savePrefs(binding.etKey.text.toString(),
binding.etValue.text.toString())
            }
        }
        binding.btnRead.setOnClickListener {
            CoroutineScope(Dispatchers.IO).launch {
                val value = readPrefs(binding.etReadingKey.text.toString())
                binding.tvValue.text = value
            }
        }
    }
```

```
    private suspend fun savePrefs(key:String, value:String) {
        val preferencesKey = stringPreferencesKey(key)
        dataStore.edit { settings ->
            settings[preferencesKey] = value
        }
    }
    private suspend fun readPrefs(key: String):String {
        val preferencesKey = stringPreferencesKey(key)
        val value: Flow<String> = dataStore.data.map {
                settings ->
            settings[preferencesKey] ?: "No value"
        }
        return value.first().toString()
    }
}
```

https://developer.android.com/topic/libraries/architecture/datastore

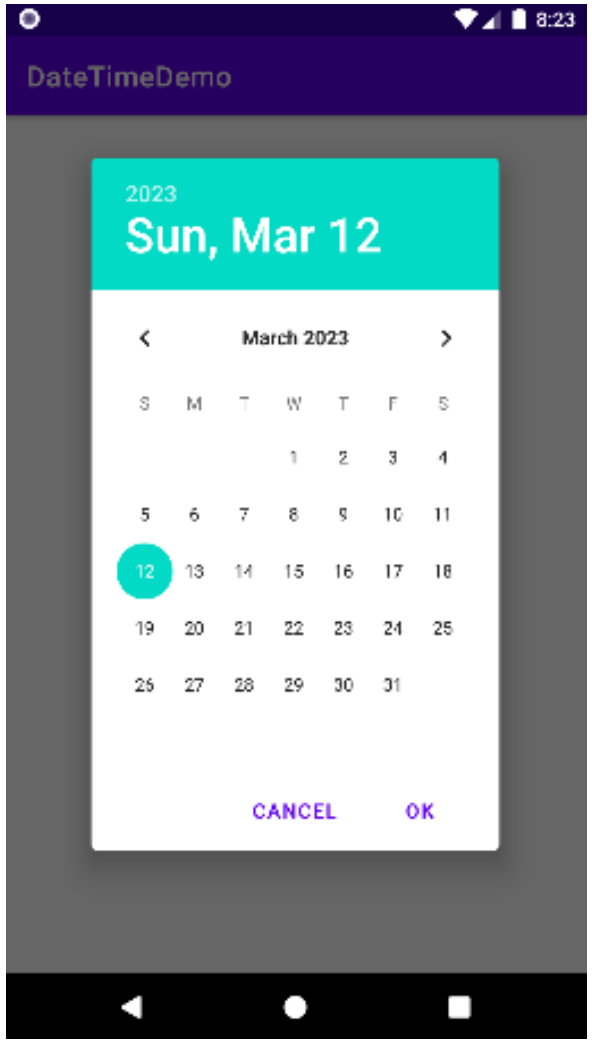
Time picker

```
val today = Calendar.getInstance()
val currentHour = today.get(Calendar.HOUR_OF_DAY)
val currentMinute = today.get(Calendar.MINUTE)
binding.btnTime.setOnClickListener {
    TimePickerDialog(this, TimePickerDialog.OnTimeSetListener { timePicker, i,
i2 ->
        // i: Giờ
        // i2: Phút
        binding.tvTime.setText("$i:$i2")
    }, currentHour, currentMinute, true).show()
}
```



Date picker

```
val today = Calendar.getInstance()
val currentYear = today.get(Calendar.YEAR)
val currentMonth = today.get(Calendar.MONTH)
val currentDayofMonth = today.get(Calendar.DAY_OF_MONTH)
binding.btnDate.setOnClickListener {
    DatePickerDialog(this, DatePickerDialog.OnDateSetListener {
datePicker, i, i2, i3 ->
        // i: Năm, i2: tháng, i3: ngày
        binding.tvDate.setText("$i3/${i2+1}/$i")
    },currentYear,currentMonth,currentDayofMonth).show()
}
```



Spinner

```
val options = arrayOf("Option 1", "Option 2", "Option 3")
val adapter = ArrayAdapter(this, android.R.layout.simple_spinner_item, options)
adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item)
binding.mySpinner.adapter = adapter

binding.mySpinner.onItemSelectedListener = object :AdapterView.OnItemSelectedListener{
    override fun onItemSelected(p0: AdapterView<*>?, p1: View?, position: Int, p3: Long) {
        val selected = options[position]
        Toast.makeText(this@MainActivity, "Your selected: $selected", Toast.LENGTH_SHORT).show()
    }

    override fun onNothingSelected(p0: AdapterView<*>?) {
    }
}
```

Custom Spinner (spinner_item.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:gravity="center_vertical"
    android:layout_height="match_parent">
    <ImageView
        android:id="@+id/iv_spinner"
        android:layout_width="50dp"
        android:layout_height="50dp"
        android:src="@drawable/cupcake"
        />
    <TextView
        android:id="@+id/tv_spinner"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="CupCake"
        android:textSize="18sp"
        android:layout_marginStart="10dp"
        android:textStyle="bold"
        />
</LinearLayout>
```

Custom Spinner

```
class AndroidVersionAdapter(val context: Context, val models: ArrayList<AndroidVersion>) : BaseAdapter() {
    override fun getCount(): Int {
        return models.size
    }

    override fun getItem(p0: Int): Any? {
        return null
    }

    override fun getItemId(p0: Int): Long {
        return 0
    }

    override fun getView(i: Int, view: View?, viewGroup: ViewGroup?): View {
        val view = LayoutInflater.from(context).inflate(R.layout.spinner_item, null)
        val icon = view.findViewById<View>(R.id.iv_spinner) as ImageView?
        val name = view.findViewById<View>(R.id.tv_spinner) as TextView?
        icon!!.setImageResource(models[i].image)
        name!!.text = models[i].heading
        return view
    }
}
```

Recycler View (text_row_item.xml)

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginLeft="20dp"
    android:layout_marginRight="20dp"
    android:gravity="center_vertical">

    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text=""/>
</FrameLayout>
```

https://developer.android.com/develop/ui/views/layout/recyclerview

Recycler View

```
class UserAdapter(private val dataSet: ArrayList<User>) :
    RecyclerView.Adapter<UserAdapter.ViewHolder>() {

    /**
     * Provide a reference to the type of views that you are using
     * (custom ViewHolder)
     */
    class ViewHolder(view: View) : RecyclerView.ViewHolder(view) {
        val textView: TextView

        init {
            // Define click listener for the ViewHolder's View
            textView = view.findViewById(R.id.textView)
        }
    }

    // Create new views (invoked by the layout manager)
    override fun onCreateViewHolder(viewGroup: ViewGroup, viewType:
Int): ViewHolder {
        // Create a new view, which defines the UI of the list item
        val view = LayoutInflater.from(viewGroup.context)
            .inflate(R.layout.text_row_item, viewGroup, false)

        return ViewHolder(view)
    }
```

```
lateinit var mlistener: OnItemClickListener

    interface OnItemClickListener {
        fun onItemClick(usr: User)
    }

    fun SetOnItemClickListener(listener: OnItemClickListener){
        mlistener = listener
    }
    // Replace the contents of a view (invoked by the layout manager)
    override fun onBindViewHolder(viewHolder: ViewHolder, position: Int) {

        // Get element from your dataset at this position and replace the
        // contents of the view with that element
        viewHolder.textView.text = dataSet[position].username
        viewHolder.textView.setOnClickListener { mlistener.onItemClick(dataSet[position])
}
    }

    // Return the size of your dataset (invoked by the layout manager)
    override fun getItemCount() = dataSet.size

}
```

Recycler View

```
binding.lvUsers.layoutManager =
LinearLayoutManager(this,LinearLayoutManager.VERTICAL, false)


var adapter = UserAdapter(listUsers)


binding.lvUsers.adapter = adapter


adapter.SetOnItemClickListener(object:UserAdapter.OnItemClickListener{
    override fun onItemClick(user: User) {

    }
})
```

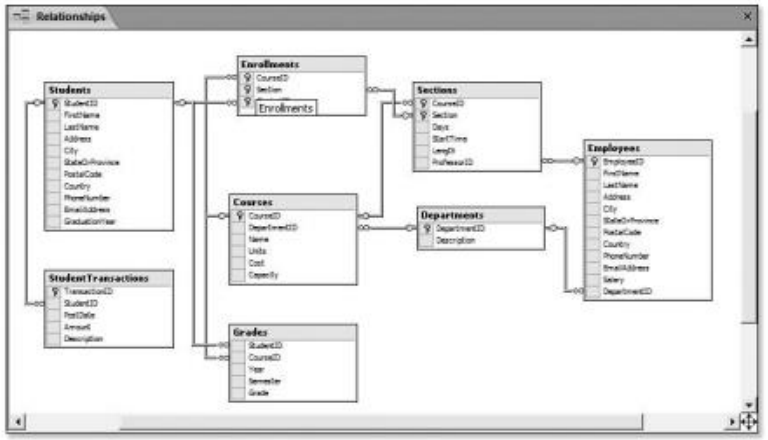
Databases and SQL

Tuần 10

What is a database?

- **relational database**: A method of structuring data as **tables** associated to each other by shared attributes.
- a table **row** corresponds to a unit of data called a record; a **column** corresponds to an attribute of that record

- relational databases typically use Structured Query Language (**SQL**) to define, manage, and search data



Why use a database?

- powerful: can search, filter, combine data from many sources
- fast: can search/filter a database very quickly compared to a file
- big: scale well up to very large data sizes
- safe: built-in mechanisms for failure recovery (transactions)
- multi-user: concurrency features let many users view/edit data at same time
- abstract: layer of abstraction between stored data and app(s)
- common syntax: database programs use same SQL commands

Some database software

- **Oracle**
- Microsoft
  - **SQL Server** (powerful)
  - **Access** (simple)
- **PostgreSQL**
  - powerful/complex free open-source database system
- **SQLite**
  - transportable, lightweight free open-source database system
- **MySQL**
  - simple free open-source database system
  - many servers run "LAMP" (Linux, Apache, MySQL, and PHP)
  - Wikipedia is run on PHP and MySQL

Example database: school

id	name	email
123	Bart	bart@fox.com
456	Milhouse	milhouse@fox.com
888	Lisa	lisa@fox.com
404	Ralph	ralph@fox.com

students

id	name
1234	Krabappel
5678	Hoover
9012	Stepp

teachers

id	name	teacher_id
10001	Computer Science 142	1234
10002	Computer Science 143	5678
10003	Computer Science 190M	9012
10004	Informatics 100	1234

courses

student_id	course_id	grade
123	10001	B-
123	10002	C
456	10001	B+
888	10002	A+
888	10003	A+
404	10004	D+

grades

SQL

```
SELECT name FROM cities WHERE id = 17;
INSERT INTO countries VALUES ('SLD', 'ENG', 'T', 100.0);
```

- **Structured Query Language** (SQL): a language for searching and updating a database
  - a standard syntax that is used by all database software *(with minor incompatibilities)*
  - generally case-insensitive

- a **declarative language**: describes what data you are seeking, not exactly how to find it

SQL

```
SELECT column(s) FROM table WHERE condition;

SELECT name, population FROM cities
                        WHERE country_code = "FSM";
```

- searches a database and returns a set of results
  - column name(s) after SELECT filter which parts of rows are returned
  - table and column names are **case-sensitive**
  - `SELECT DISTINCT` removes any duplicates
  - `SELECT *` keeps all columns
- WHERE clause filters out rows based on columns' data values
  - in large databases, WHERE clause is critical to reduce result set size

WHERE clauses

```
SELECT name, gnp FROM countries WHERE gnp > 2000000;

SELECT * FROM cities WHERE code = 'USA'
                     AND population >= 2000000;

SELECT code, name, population FROM countries
WHERE  name LIKE 'United%';
```

- WHERE clause can use the following operators:

  =, >, >=, <, <=

  <> : not equal (some systems support !=)

  BETWEEN ***min*** AND ***max***

  LIKE ***pattern*** *(put % on ends to search for prefix/suffix/substring)*

  IN (***value***, ***value***, ..., ***value***)

  condition1 AND ***condition2*** ; ***condition1*** OR ***condition2***

ORDER BY, LIMIT

```
SELECT code, name, population FROM countries
WHERE name LIKE 'United%' ORDER BY population;

SELECT * FROM countries ORDER BY population DESC, gnp;

SELECT name FROM cities WHERE name LIKE 'K%' LIMIT 5;
```

- ORDER BY sorts in ascending (default) or descending order
  - can specify multiple orderings in decreasing order of significance

- LIMIT gets first N results of the query
  - useful as a sanity check to make sure query doesn't return 10^7 rows

JOIN

```
SELECT column(s) FROM table1 name1
                 JOIN table2 name2 ON condition(s)
                 ...
                 JOIN tableN nameN ON condition(s)
                 WHERE condition;
```

```
SELECT name, course_id, grade
FROM students s
JOIN grades g ON s.id = g.student_id
WHERE s.name = 'Bart';
```

- JOIN combines related records from two or more tables
  - ON clause specifies which records from each table are matched
  - rows are often linked by their key columns ('id')
  - joins can be tricky to understand; out of scope of this course

Create/delete a database; CRUD

```
CREATE DATABASE name;
DROP DATABASE name;

CREATE DATABASE warcraft;
```

- Must first create a database and add one or more tables to it.

- Most apps/sites do four general tasks with data in a database:
  - **C**reate new rows
  - **R**ead existing data
  - **U**pdate / modify values in existing rows
  - **D**elete rows

Creating tables

```
CREATE TABLE IF NOT EXISTS name (
  columnName type constraints,
  ...
  columnName type constraints
);
DROP TABLE name;

CREATE TABLE students (
  id INTEGER,
  name VARCHAR(20),
  email VARCHAR(32),
  password VARCHAR(16)
);
```

BOOLEAN	either TRUE or FALSE
INTEGER	32-bit integer
DOUBLE	real number
VARCHAR(***length***)	string up to given length
ENUM(***val***, ..., ***val***)	a fixed set of values
DATE, TIME, DATETIME	timestamps (common value: NOW())
BLOB	binary data

Table column constraints

```
CREATE TABLE students (
  id INTEGER UNSIGNED NOT NULL PRIMARY KEY AUTO_INCREMENT,
  name VARCHAR(20) NOT NULL,
  email VARCHAR(32),
  password VARCHAR(16) NOT NULL DEFAULT "12345"
);
```

- NOT NULL: empty value not allowed in any row for that colum
- PRIMARY KEY / UNIQUE: no two rows can have the same value
- DEFAULT ***value***: if no value is provided, use the given default
- AUTO_INCREMENT: default value is the last row's value plus 1
  - (usually used for ID column)
- UNSIGNED: don't allow negative numbers (INTEGER only)

INSERT and REPLACE

```
INSERT INTO table (columnName, ..., columnName)
VALUES (value, value, ..., value);

REPLACE INTO table (columnName, ..., columnName)
VALUES (value, value, ..., value);


INSERT INTO students (name, email)
VALUES ("Lewis", "lewis@fox.com");

REPLACE INTO students (id, name, value)
VALUES (789, "Martin", "prince@fox.com");
```

- some columns have default or automatic values (such as IDs)
- omitting them from the INSERT statement uses the defaults
- REPLACE is like INSERT but modifies an existing row

UPDATE

```
UPDATE table
SET column1 = value1,
    ...,
    columnN = valueN
WHERE condition;

UPDATE students
SET email = "lisasimpson@gmail.com"
WHERE id = 888;
```

- modifies an existing row(s) in a table
- Be careful! If you omit WHERE clause, it modifies ALL rows

DELETE

```
DELETE FROM table
WHERE condition;

DELETE FROM students
WHERE id = 888;
```

- removes existing row(s) in a table
- can be used with other syntax like LIMIT, LIKE, ORDER BY, etc.
- Be careful! If you omit WHERE clause, it deletes ALL rows

SQLite - Init

```
class MyDB(context: Context): SQLiteOpenHelper(context, "users",null,1) {
    override fun onCreate(p0: SQLiteDatabase?) {
        p0?.execSQL("create table users(id integer primary key autoincrement not null, usr text, pwd text)")
        p0?.execSQL("insert into users(usr,pwd) values ('abc1@gmail.com','123456')")
        p0?.execSQL("insert into users(usr,pwd) values ('abc2@gmail.com','123457')")
    }

    override fun onUpgrade(p0: SQLiteDatabase?, p1: Int, p2: Int) {
        TODO("Not yet implemented")
    }
}
```

```
data class User(val id: Int, val username: String, val password: String)
```

SQLite - Query

```
val helper = MyDB(applicationContext)
db =  helper.readableDatabase
var rs = db.rawQuery("select * from users", null)
listUsers = ArrayList<User>()
while(rs.moveToNext()){
    listUsers.add(User(rs.getInt(0),rs.getString(1),rs.getString(2)))
}
```

SQLite - Insert

```
var cv = ContentValues()
cv.put("usr", binding.etUsername.text.toString())
cv.put("pwd", binding.etUsername.text.toString())
db.insert("users", null, cv)
var rs = db.rawQuery("select * from users", null)
rs.moveToLast()
listUsers.add(User(rs.getInt(0), rs.getString(1),rs.getString(2)))
adapter.notifyDataSetChanged()
```

SQLite - Delete

```
db.delete("users","id=?", arrayOf(user.id.toString()))
var rs = db.rawQuery("select * from users", null)
listUsers.clear()
while(rs.moveToNext()){
    listUsers.add(User(rs.getInt(0),rs.getString(1),rs.getString(2)))
}
adapter.notifyDataSetChanged()
```

SQLite - Update

```
var cv = ContentValues()
cv.put("usr", binding.etUsername.text.toString())
cv.put("pwd", binding.etUsername.text.toString())
db.update("users", cv, "id=?", arrayOf(selectUser.id.toString()))
var rs = db.rawQuery("select * from users", null)
listUsers.clear()
while(rs.moveToNext()){
    listUsers.add(User(rs.getInt(0),rs.getString(1),rs.getString(2)))
}
adapter.notifyDataSetChanged()
```

Bài tập

Tạo ứng dụng quản lý danh bạ cục bộ sử dụng SQLite, có các như năng:

- Back: Xem danh bạ trước đó
- Next: Xem danh bạ kế tiếp
- Add: Thêm danh bạ mới
- Save: Lưu danh bạ
- Delete: Xóa danh bạ