Realm Database Cheatsheet

A concise guide to using Realm, covering schema definition, CRUD operations, queries, and relationships.

Core Concepts & Setup

Realm Fundamentals

Realm: A mobile database solution that offers an alternative to SQLite and Core Data. It's designed for speed and ease of use.

Key Features:

- **Real-time:** Data changes are immediately reflected.
- **Cross-platform:** Supports multiple platforms (iOS, Android, React Native, etc.).
- **Object-oriented:** Data is represented as objects.

Data Model: Realm uses a schema to define the structure of your data. Models are defined as classes.

Installation (Swift): Add `realm-swift` to your `Podfile` or use Swift Package Manager.

Importing Realm:

```
import RealmSwift
```

Configuration

Default Realm	The default Realm is suitable for most basic use cases. It stores data in the app's default location.
Custom Realm Configuration	Use `Realm.Configuration` to customize Realm's behavior, like specifying a different file path or encryption key.
In-Memory Realm	Useful for testing. Data is not persisted to disk. `Realm.Configuration.defaultConfiguration = Realm.Configuration(inMemoryIdentifier: "MyInMemoryRealm")`

Error Handling

Realm throws exceptions for various errors. Wrap Realm operations in `do-catch` blocks to handle them.

Common Errors:

- **Invalid schema:** Incorrect property types or missing primary keys.
- **Migration required:** Schema changes necessitate a migration.

Example:

```
do {
    let realm = try Realm()
} catch {
    print("Error initializing Realm: (error)")
}
```

Defining Realm Models

Basic Model Definition

Realm models are defined as classes that inherit from `Object`.

Properties must be declared with the `@objc dynamic var` prefix to enable Realm's change tracking.

Example:

```
class Dog: Object {
    @objc dynamic var name = ""
    @objc dynamic var age = 0
}
```

Supported Data Types

`Int`	Integer numbers.
`Double`, `Float`	Floating-point numbers.
`String`	Textual data.
`Bool`	Boolean values (true/false).
`Date`	Date and time values.
`Data`	Binary data.

Optional Properties

Properties can be declared as optional using `?`.

Optional properties can store `nil` values.

Example:

```
class Person: Object {
    @objc dynamic var name: String? = nil
}
```

Ignored Properties

Properties marked with `@objc ignore` are not persisted to the Realm file.

Useful for temporary or calculated values.

Example:

```
class Rectangle: Object {
    @objc dynamic var width = 0
    @objc dynamic var height = 0
    @objc ignore var area: Int {
        return width * height
    }
}
```

CRUD Operations

Creating Objects

Create instances of your Realm model classes and add them to the Realm.

Example:

```
do {
    let realm = try Realm()
    try realm.write {
        let dog = Dog()
        dog.name = "Buddy"
        dog.age = 3
        realm.add(dog)
    }
} catch {
    print("Error creating object:
(error)")
}
```

Reading Objects

Use Realm queries to retrieve objects.

Example:

```
do {
    let realm = try Realm()
    let dogs = realm.objects(Dog.self)
    for dog in dogs {
        print("Dog name: (dog.name),
age: (dog.age)")
    }
} catch {
    print("Error reading objects:
(error)")
}
```

Updating Objects

Update objects within a write transaction.

Example:

```
do {
    let realm = try Realm()
    let dog =
realm.objects(Dog.self).first
    try realm.write {
        dog?.age = 4
    }
} catch {
    print("Error updating object:
(error)")
}
```

Deleting Objects

Delete objects within a write transaction.

Example:

```
do {
    let realm = try Realm()
    let dog =
realm.objects(Dog.self).first
    try realm.write {
        if let dogToDelete = dog {
            realm.delete(dogToDelete)
        }
    }
} catch {
    print("Error deleting object:
(error)")
}
```

Querying Realm Data

Basic Queries

Realm uses a query language similar to NSPredicate.

Use `realm.objects(YourModel.self).filter("your_query")` to filter results.

Example:

```
let youngDogs =
realm.objects(Dog.self).filter("age <
5")
```

Common Query Operators

`=`	Equals.
`!=`	Not equals.
`>`	Greater than.
`<`	Less than.
`>=`	Greater than or equal to.
`<=`	Less than or equal to.
`BEGINSWITH`	String starts with.
`ENDSWITH`	String ends with.
`CONTAINS`	String contains.
`LIKE`	String matches a wildcard pattern.

Compound Predicates

Combine predicates using `AND`, `OR`, and `NOT`.

Example:

```
let query = "age > 2 AND name BEGINSWITH
'B'"
let results =
realm.objects(Dog.self).filter(query)
```

Sorting Results

Use `sorted(byKeyPath:ascending:)` to sort results.

Example:

```
let sortedDogs =
realm.objects(Dog.self).sorted(byKeyPath
: "age", ascending: true)
```