[[Ch06 – Application design and implementation]]

# == Application design and implementation

Like all databases, CockroachDB responds to requests from application code. How an application requests and uses data has a huge bearing on application performance and scalability. In this chapter, we’ll review how an application should work with CockroachDB – including best practices for coding CockroachDB requests and transactional models.

### === Installing a programming language driver

In this section we’ll review the basics of working with CockroachDB in four of the most common programming languages, as well as a summary of support for other languages.

#### ==== Java

#### ==== NodeJS

#### ==== GoLang

#### ==== Python

#### ==== Other languages

### === Coding Best Practices

In this section, we’ll look at how best to code around the various CockroachDB driver APIs. Regardless of what driver you are using, you’ll want to avoid unnecessary database requests, creating SQL injection vulnerabilities, reducing network round trips and in other ways, avoid inefficient interactions with the database.

#### ==== Parameterised statements

#### ==== Array processing

#### ==== Connection pools

#### ==== Projections

### === Working with ORM Frameworks

While many applications work directly with CockroachDB via SQL statements embedded in application code, other applications use frameworks that avoid the direct use of SQL and instead leverage automated mapping of database tables to program objects. In this section, we’ll introduce some of the most popular and provide an example of their use.

#### ==== SQLAlchemy for Python

#### ==== Django for Python

#### ==== Java Hibernate

#### ==== Java JOOQ

#### ==== GORM for GoLang

#### ==== TypeORM for NodeJS

### === Transaction Handling

CockroachDB is a transactional system – even if you don’t explicitly leverage transactions you will experience the effects of CockroachDB transaction handling.

In this section we’ll discuss how to program against the transactional system, and how to handle transaction retries and other transactional handling patterns.

#### ==== Optimistic vs Pessimistic strategy

#### ==== Automatic transaction retries

#### ==== Client side intervention

#### ==== Time travel queries

#### ==== Deadlocks

#### ==== Nested trasnactions

#### ==== Transaction priorities