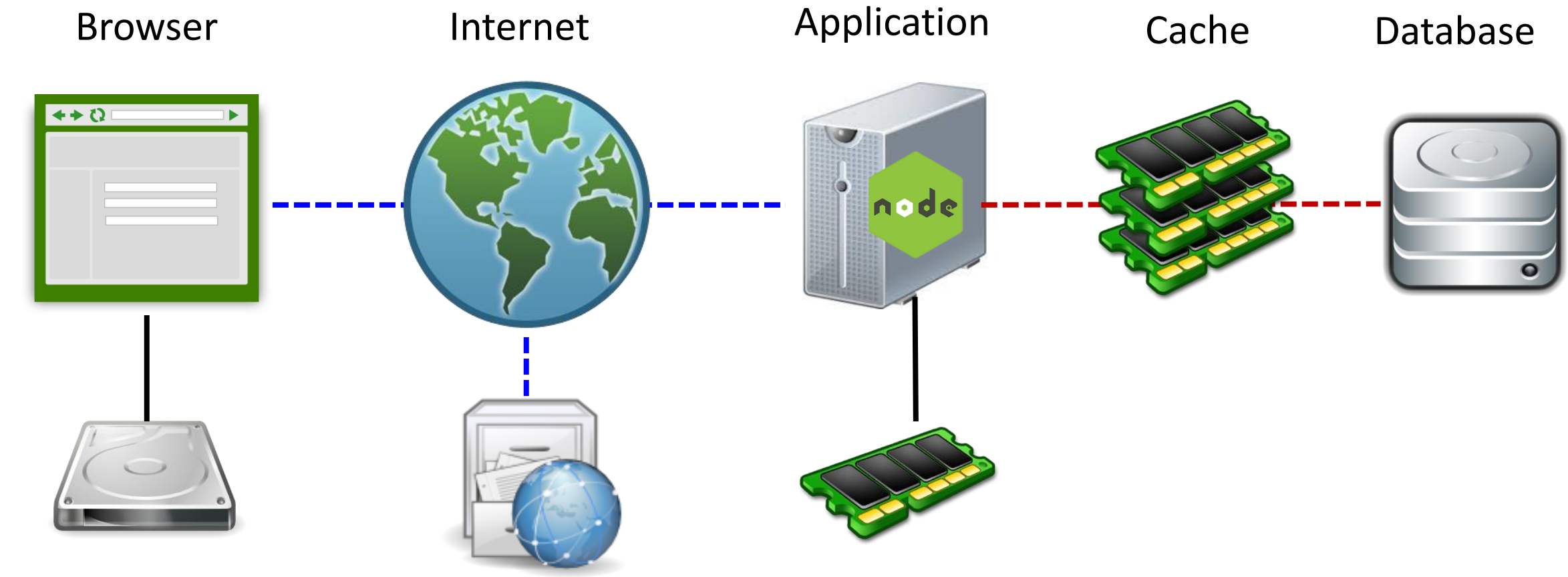
Day 21

Where Can Data be Persisted?



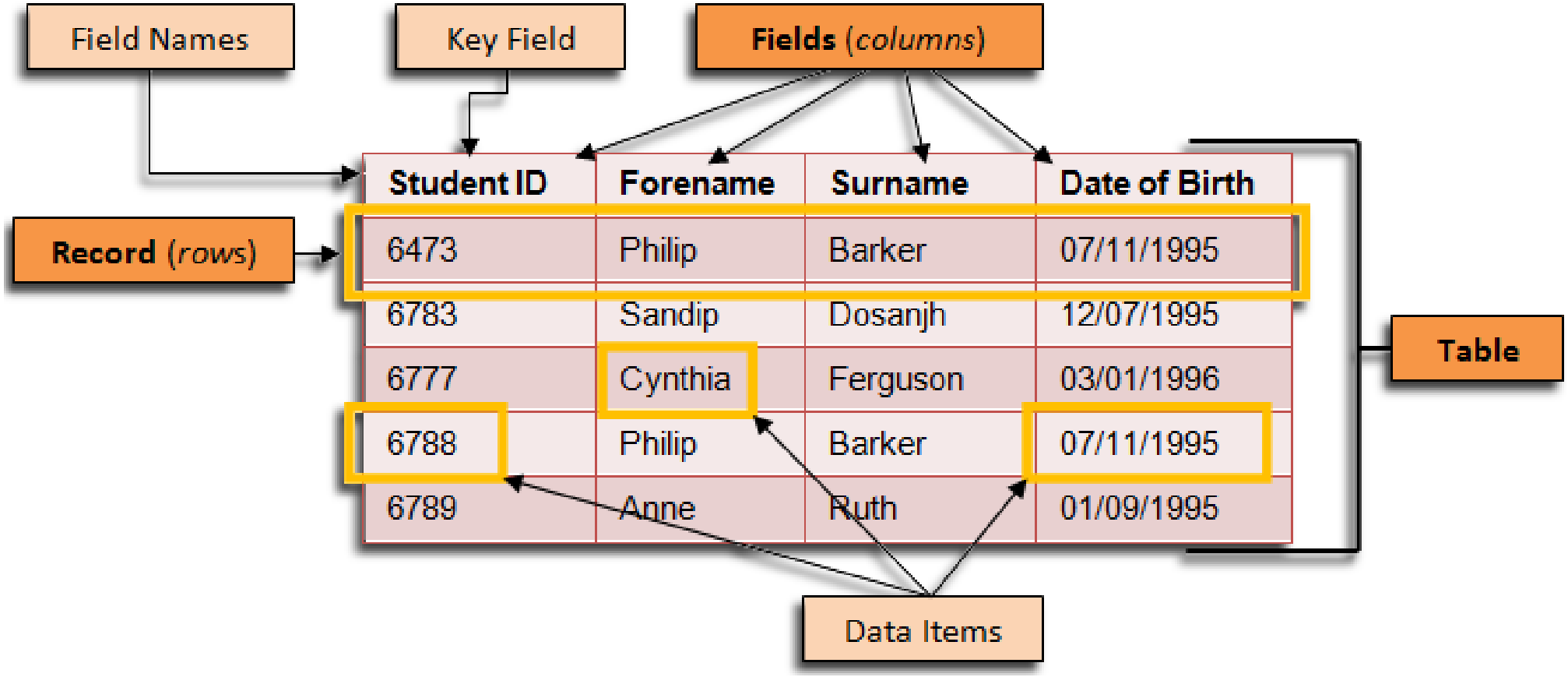
What are Databases?

- A system for storing data independent of an application
  - Consequence: a database may be access by many different applications
- Why?
  - Impose data integrity rules
  - Control access to the data
  - Manipulate the data independent of any application
  - Queried

Relational Database

- Relational database are organized like a spreadsheet
  - A table is like a sheet - storing similar data
  - A row is a record
  - Columns for storing different categories of data pertaining to a single record
- Need to know all schema (structure of the data) before storing any data in a relational database
- Relationships may existing between tables
  - Eg. A table storing authors may have multiple records (rows) in the book table
- Standard language to query and manipulate data
  - SQL - Structured Query Language
- Tables may be joined via relationships between fields
- Supports transactions - ACID properties
  - Atomicity, Consistency, Isolation, Durability

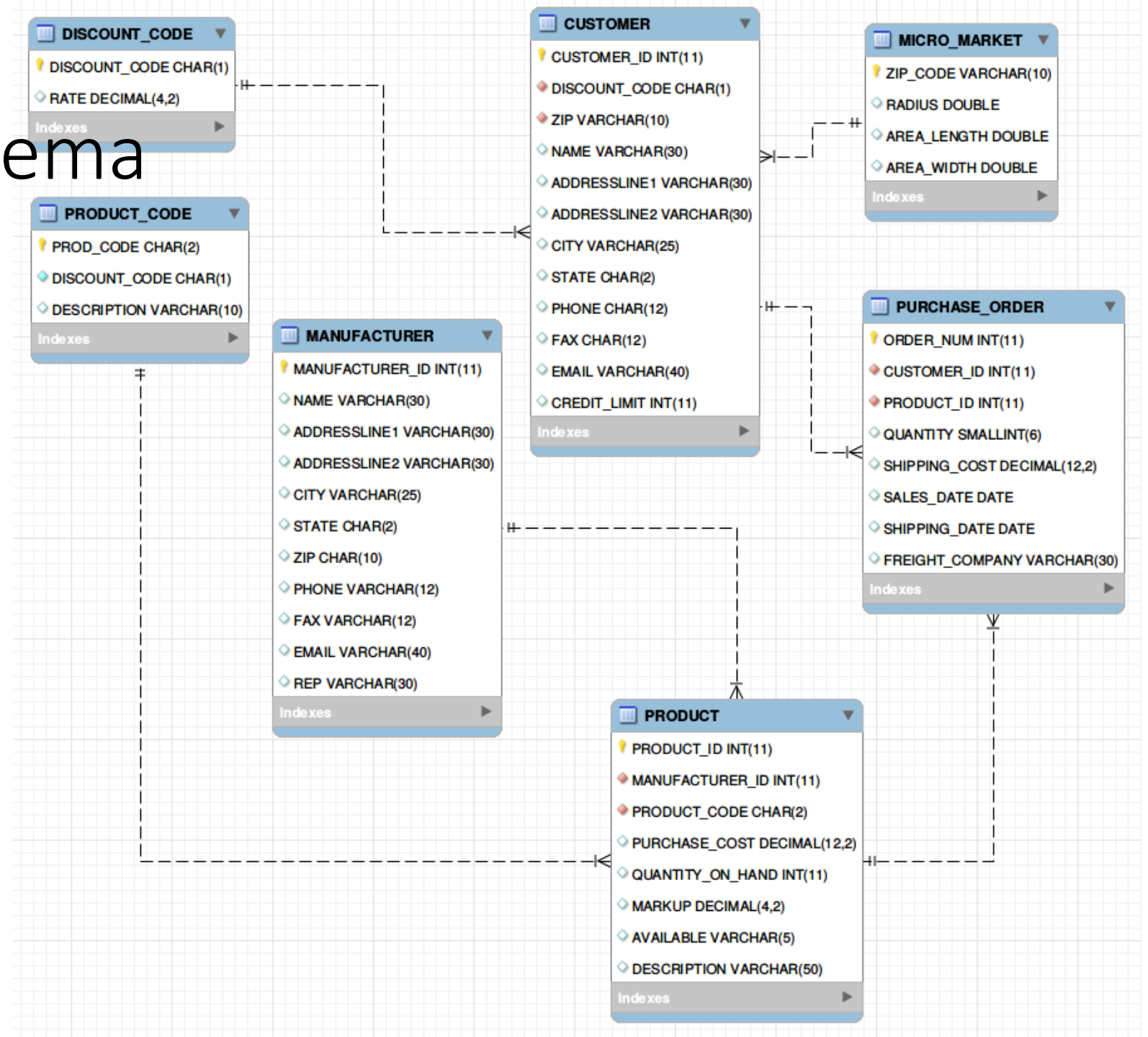
Structure of a Database Table



https://gcsecomputing.org.uk/theory/relational-databases/

Database Schema

Schema refers to the organization of data



Creating a Database

Database/schema name

```
create database leisure;

use leisure;
```

Make this the current working database

Creating a Table

Table name

Column name

Additional properties for the column

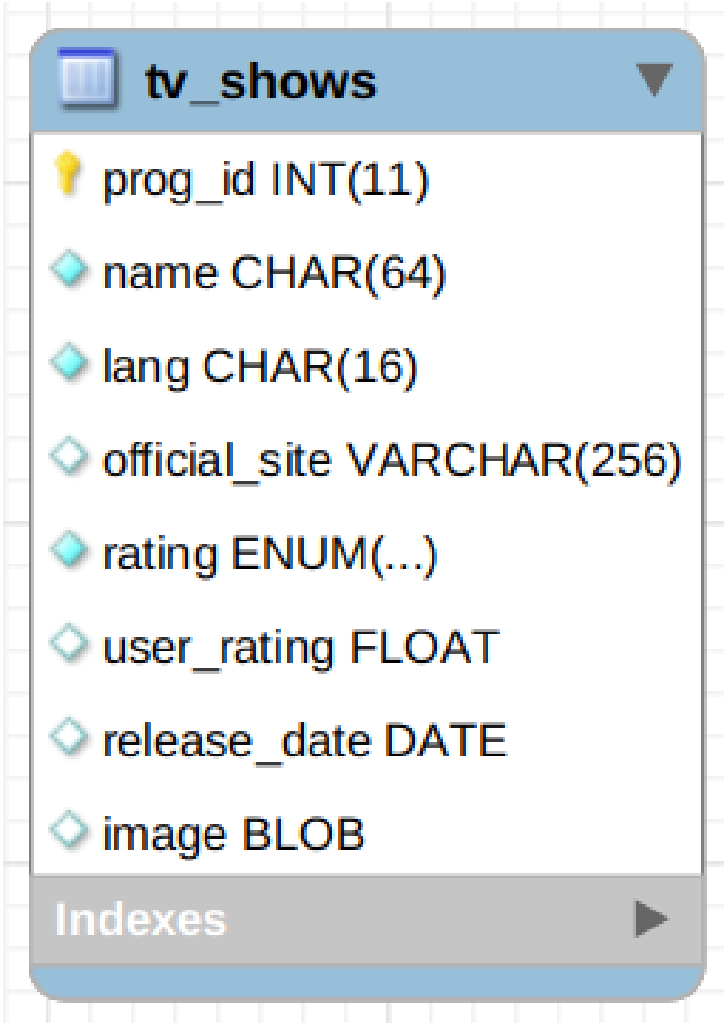
```
create table tv_shows (

    prog_id          int       not null,
    name             char(64)  not null,
    lang             char(16)  not null,
    official_site    varchar(256),
    rating           enum('G','PG', 'NC16', 'M18', 'R21') not null,
    user_rating      float default '0.0',
    release_date     date,
    image            blob,
   primary key (prog_id)
);
```

Data type of each column

The designated column to be the primary key

`tv_shows` Entity Diagram



Data Types

- Define the type of data a column can store
- Some common data type
  - Numbers - int, decimal
  - Characters - char, varchar, enum
  - Time - date, timestamp
  - Large text - text
  - Binary objects - blob
  - JSON

Primary Key

- A field from a record that is used to identify that record
  - This is the field that uniquely identifies a record
  - Eg. NRIC, product code
- Properties of a primary key field
  - Cannot be null/empty
  - The value of the field must be unique
  - Once assigned a primary key value should not be changed
  - A primary key is assigned to a newly inserted record
- Can be auto generated
  - Column type must be an integer
  - Add the `auto_increment` attribute

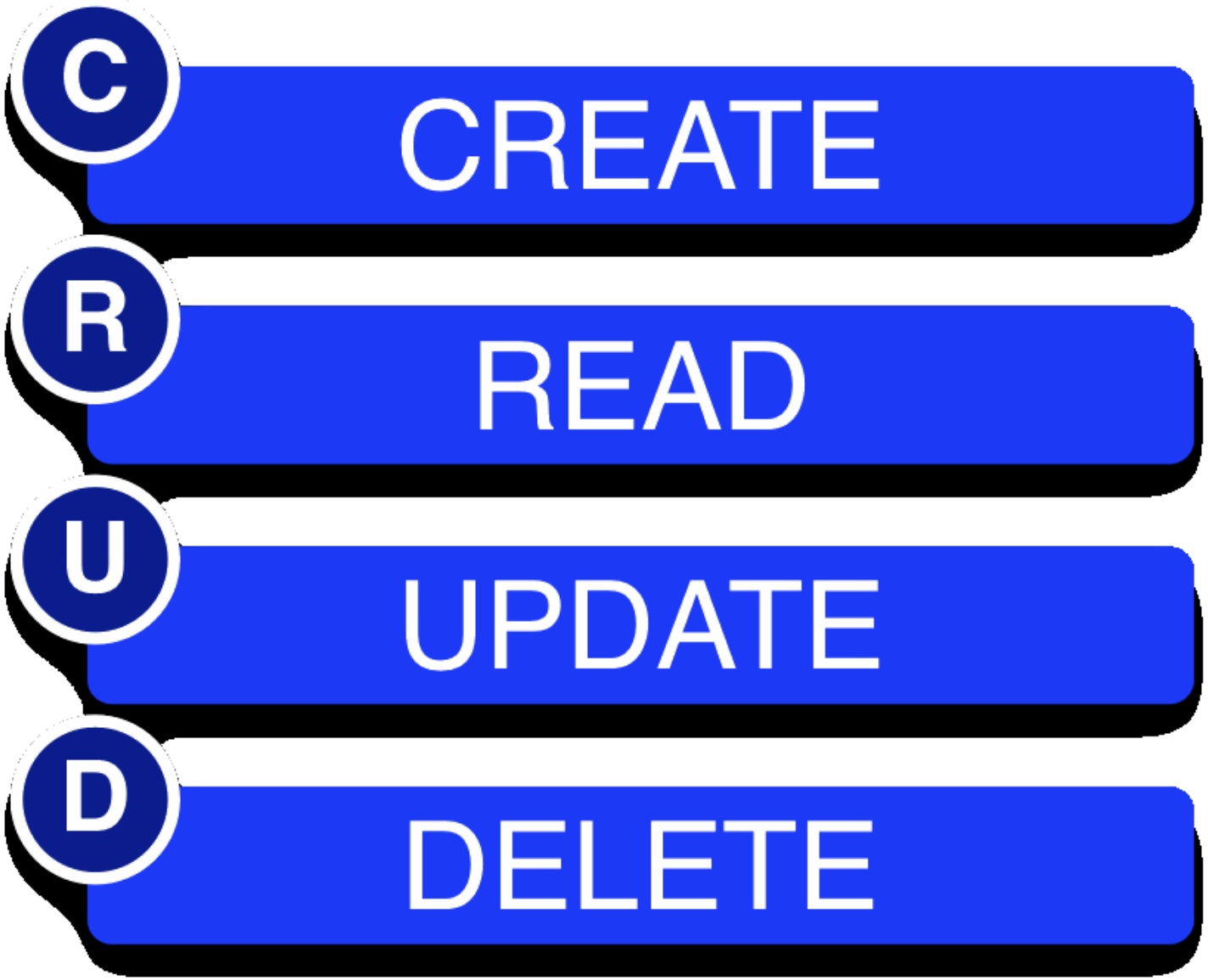
Primary Key

- Generating primary keys
  - Get the user to provide - eg. selecting a name for your Email account
  - Get the database to generate - field must be an integer
- Can create tables without primary key
  - Eg. many to many join table

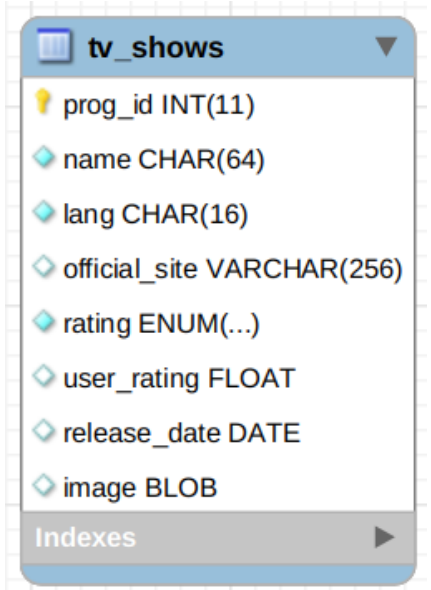
Auto generated primary key

```
create table tv_shows (
  prog_id      int(11)   not null auto_increment,
  ...
  primary key (prog_id)
)
```

CRUD



Read - SELECT



Retrieve all the records

```
select * from tv_shows;
```

Table name

All the fields

Retrieve the first 100 record

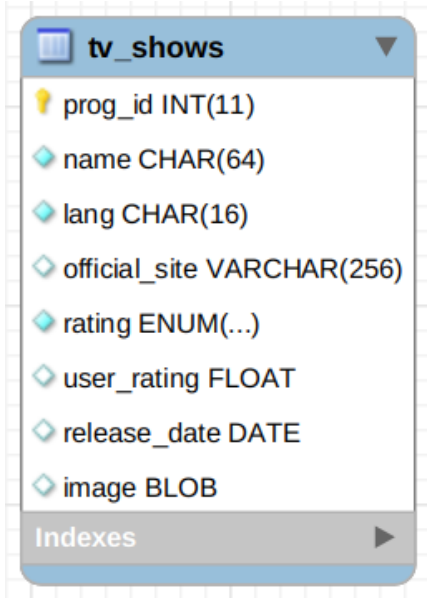
```
select * from tv_shows limit 100;
```

Retrieve only `prog_id` and `name` fields

```
select prog_id, name from tv_shows
    limit 100 offset 1000;
```

Retrieve the first 100 record starting from the 1000th

Reading Specific Data



Find all records whose `name` is `Gotham`

Provide a condition for the query

Read = as 'is'

```
select * from tv_shows where name = "Gothan";
```

Find all records whose `name` has the string `bad`

Match a pattern
Case insensitive

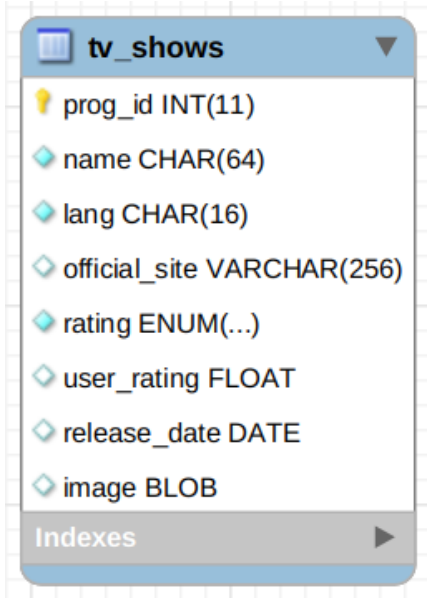
'wildcard' - any number of characters

```
select * from tv_shows where last_name like "%bad%";
```

```
select release_date from tv_shows
    where dob > "2012-01-01" and
    dob < "2012-12-31;
```

Find all TV shows release between Jan 01 till Dec 31 in 2012

Reading Specific Data




```
select * from tv_shows where ratings
    in ("M18", "R21");
```

`ratings` is matches one of the values in the given set

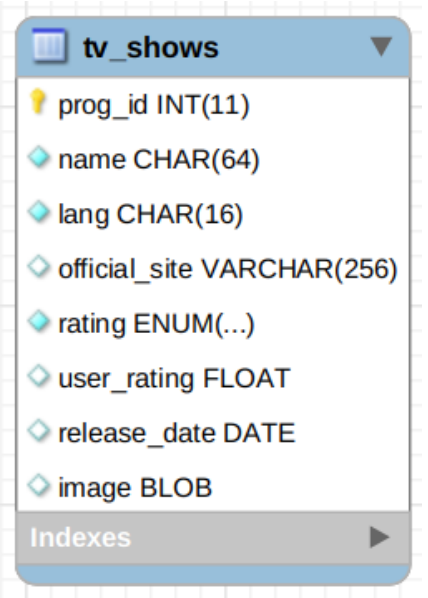
```
select name from tv_shows
    where ratings not between
     "2012-01-01" and "2012-12-31;
```

The between operator makes conditions more English like

Logical and Comparison Operators

- Logical - https://dev.mysql.com/doc/refman/8.0/en/comparison-operators.html
  - `and`
  - `or`
  - `not`
- Comparison - https://dev.mysql.com/doc/refman/8.0/en/logical-operators.html
  - > greater than
  - < less than
  - <= greater than or equal to
  - >= less than or equal to
  - <> not equals
  - = equals

Operators Example



- Comparison operators

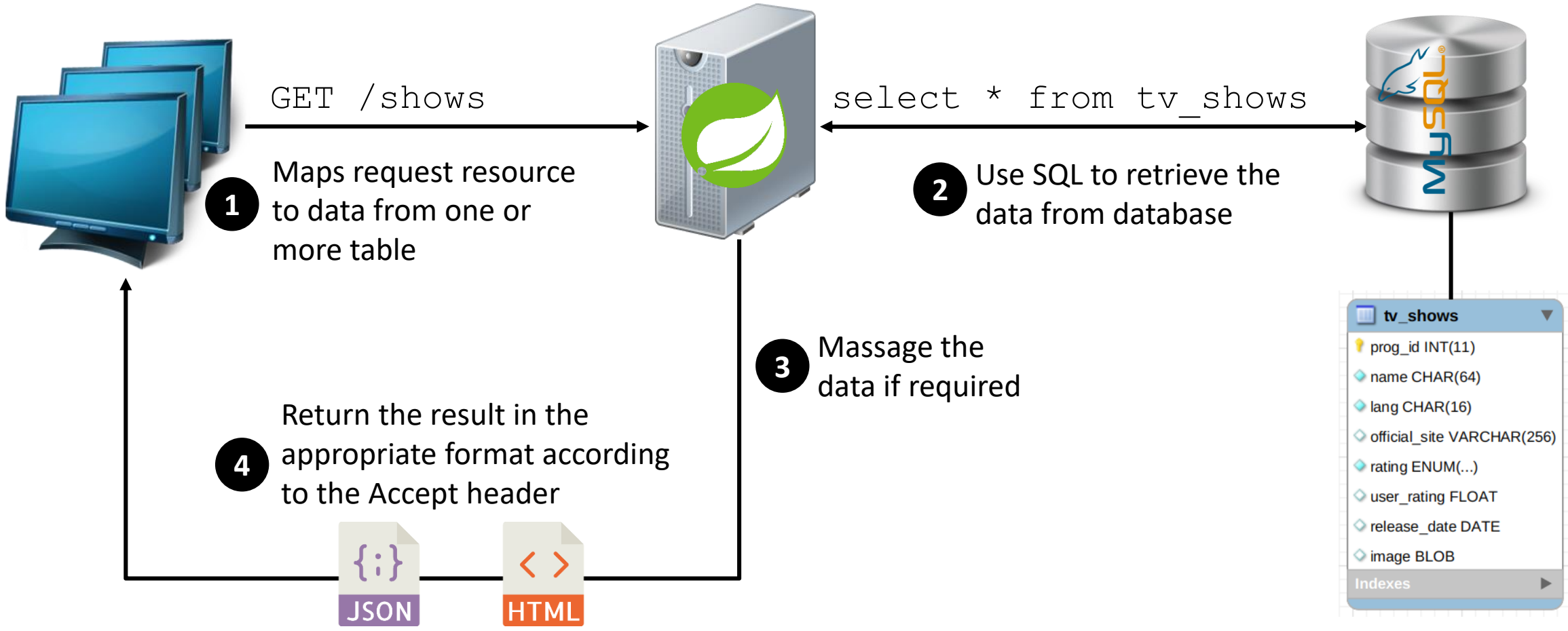
Return the first non null column for each record

```
select name
from tv_shows
where release_data between “2020-01-01” and “2022-31-12”;
```

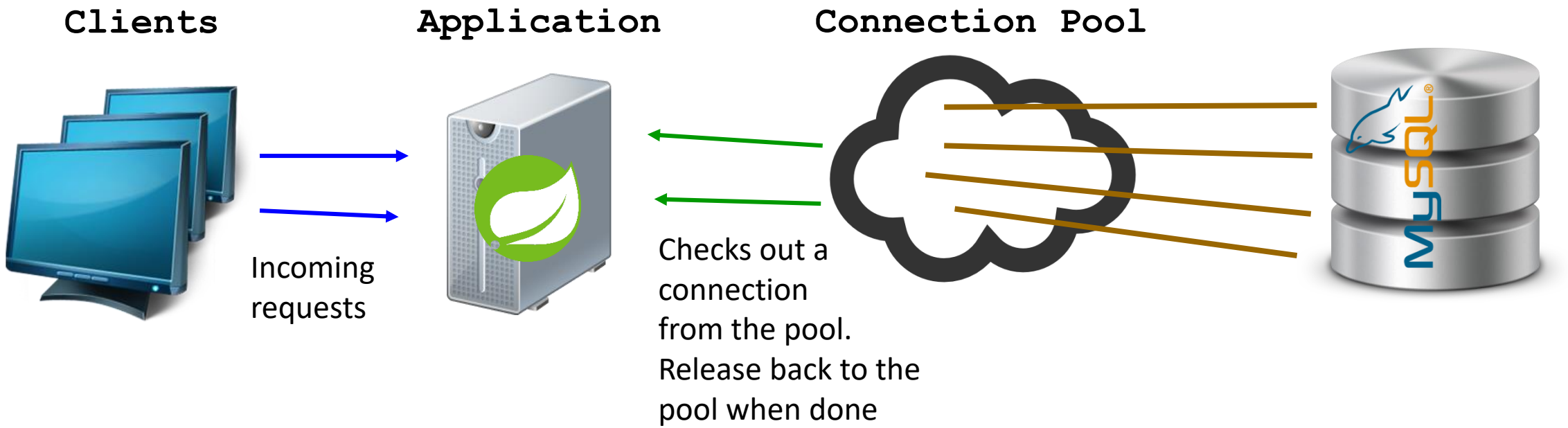
- Logical operators

```
select name
from tv_shows
where user_ratings > 5 and rating in (“M18”, “R21”);
```

Integrating Database to Spring



Connection Pool



Connecting to MySQL

Dependencies | **ADD DEPENDENCIES...** CTRL + B

Spring Boot DevTools DEVELOPER TOOLS

Provides fast application restarts, LiveReload, and configurations for enhanced development experience.

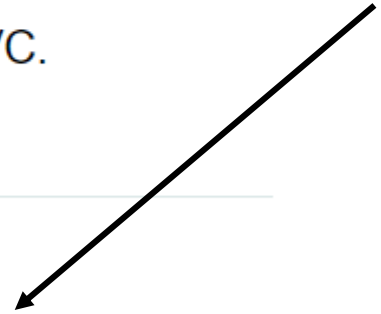
Spring Web WEB

Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

JDBC API SQL

Database Connectivity API that defines how a client may connect and query a database.

Add JDBC API dependency and MySQL Connector/J (from mvnrepository.com`)



Configuring JDBC Connection Pool

application.properties

```
spring.datasource.url=jdbc:mysql://localhost:3306/leisure
spring.datasource.username=fred
spring.datasource.password=fred

spring.datasource.hikari.connectionTimeout=30000
spring.datasource.hikari.idleTimeout=600000
spring.datasource.hikari.minimumIdle=2
spring.datasource.hikari.maximumPoolSize=8
```

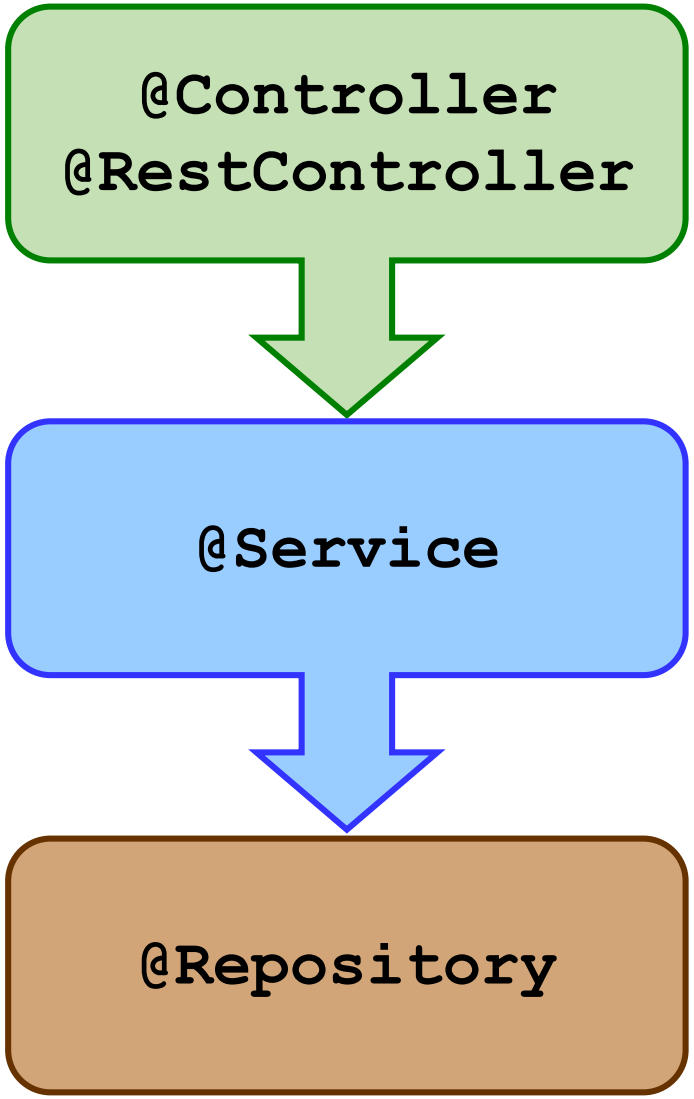
JDBC database connection string

Database user

Connection pool characteristics

@Repository

- Annotates a classes as a repository
  - Indicates that these classes are at the persistent layer
- Spring provides different persistent options
  - `JdbcTemplate` - directly accessing the records and fields
  - `CrudRepository` - maps records to Java objects. Repository automatically implements CRUD operation
  - `JpaRepository` - build on JPA ORM



Example - @Repository Example

```
@Repository
public class TVShowRepository {

    @Autowired
    private JdbcTemplate template;

    public List<TVShow> getTvShows(final int limit, final int offset) {
        final List<TVShow> result = new LinkedList<>();

        final SqlRowSet rs = template.queryForRowSet(
            "select * from tv_shows limit ? offset ?", limit, offset);

        while (rs.next()) {
            TVShow tv = new TVShow();
            tv.setProgramId(rs.getInt("prog_id");
            tv.setName(rs.getString("name"));
            ...
            result.add(tv);
        }
        return (Collections.unmodifiableList(result));
    }
}
```

Annotate this class as a repository (data access object)

Inject an instance of `JdbcTemplate`. Must be configured to a specific database

Example - @Repository Example

```
@Repository
public class TVShowRepository {

    @Autowired
    private JdbcTemplate template;

    public List<TVShow> getTvShows(final int limit, final int offset) {
        final List<TVShow> result = template.query(
            "select * from tv_shows limit ? offset ?",
            (rs, int) -> {
                TVShow tv = new TVShow();
                tv.setProgramId(rs.getInt("prog_id");
                tv.setName(rs.getString("name"));
                ...
            },
            limit, offset);

        return (Collections.unmodifiableList(result));
    }
}
```

Map records to object with RowMapper

Performing Query with Prepared Statement

```
final SqlRowSet rs = template.queryForRowSet(
   "select * from tv_shows limit ? offset ?", limit, offset
);
```

```
final SqlRowSet rs = template.queryForRowSet(
   String.format("select * from tv_shows limit %d offset %d", limit, offset")
);
```

Never do this

- Use prepared statement to construct query
  - **?** as placeholder for values
- Do not use string concatenation
  - Can result in SQL injection

Performing Query with Prepared Statement

```
final float userRating = 4f;
final String rating = "M18";

final SqlRowSet rs = template.queryForRowSet(
    "select * from tv_shows where user_rating > ? and rating = ?"
    , userRating, rating
);
```

`user_rating` is float in the schema
`rating` is char in the schema

Parameters for the prepared statement are float and string respectively

- Use the correct type for the prepared statement's parameters
- Might get error or unexpected result

Reading the Result

- `SqlRowSet` holds the result from a `queryForRowSet()`
- Navigating
  - `next()` - moves the cursor to the next record,
    - Returns true if the next record exists, false otherwise; end of results
    - Must call `next()` before reading the first record
  - `previous()` - moves the cursor back a record, returns boolean as per `next()`
- Reading fields from record with `get<data type>(<column name>)`
  - Eg. `rs.getString("email")` - returns the value in email field of the current record
  - Will return unexpected data or throws exception if read with incorrect data type
- `SqlRowSet` holds the data in memory, can cause the JVM to throw `OutOfMemoryErrorException` if may request holding large `SqlRowSet`

Example - Iterating over the Result

Return a `RowSet` object to hold the result

Prepared SQL statement with parameters

```
final SqlRowSet rs = template.queryForRowSet(
      "select * from tv_shows limit ? offset ?", limit, offset);

while (rs.next()) {
    TVShow tv = new TVShow();
    tv.setProgramId(rs.getInt("prog_id");
    tv.setName(rs.getString("name"));
    ...
}
```

Iterate through the row set

Use the appropriate `getXXX()` method to read the fields

Instantiate an instance of the table's object - to hold the record (data transfer object)

Three Tier Application

Presentation tier

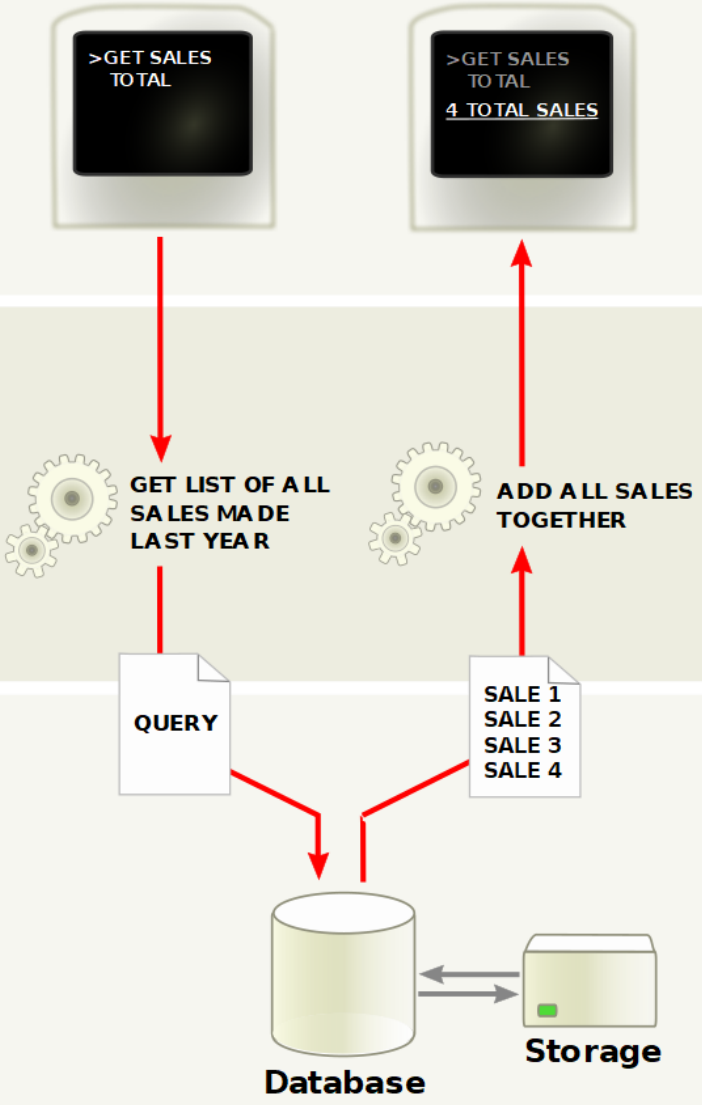
The top-most level of the application is the user interface. The main function of the interface is to translate tasks and results to something the user can understand.

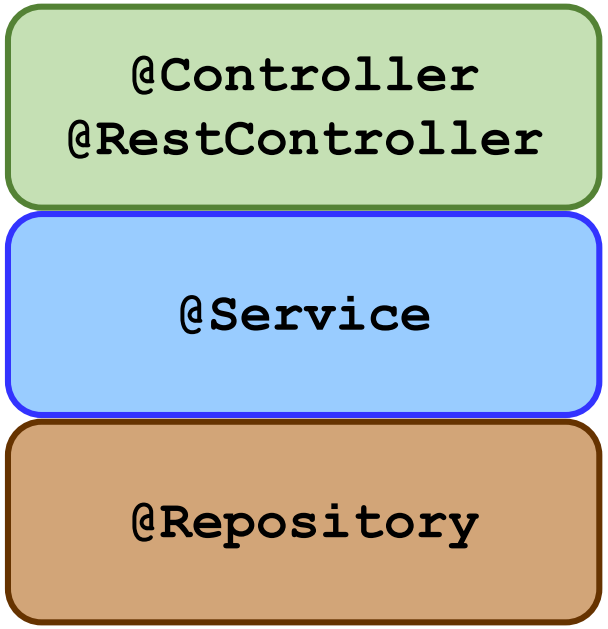
Logic tier

This layer coordinates the application, processes commands, makes logical decisions and evaluations, and performs calculations. It also moves and processes data between the two surrounding layers.

Data tier

Here information is stored and retrieved from a database or file system. The information is then passed back to the logic tier for processing, and then eventually back to the user.





Handling a Request - Controller

```
GET /tvshow/1
GET /tvshow/2
```

```
@RestController
@RequestMapping(path="/tvshow" , produces=MediaType.APPLICATION_JSON_VALUE)
public class TvShowController {
    @Autowired private TVShowService tvSvc;

    @GetMapping("{tvId}")
    public ResponseEntity<String> getBook(@PathVariable Integer tvId) {
        Optional<TVShow> opt = tvSvc.findShowById(tvId);

        if (opt.isEmpty())
            return ResponseEntity.status(404).body(
                Json.createObjecBuilder().add("message", "Cannot find " + tvId")
                    .build().toString());

        return ResponseEntity.ok(opt.get().toJSON().toString());
    }
}
```

Handling a Request - Service

```
@Service
public class TVShowServie {
    @Autowired
    private TVShowRepository tvShowRepo;

    public Optional<TVShow> findShowById(final Integer tvId) {
        SqlRowSet rs = tvShowRepo.get(tvId);
        if (rs.first())
            return Optional.of(TVShow.populate(rs));
        return Optional.empty();
    }
}
```

Can return `TVShow` or `SqlRowSet`
The latter provider more flexibility at the expense of 'leaky abstraction' but more flexible
https://en.wikipedia.org/wiki/Leaky_abstraction

Handling a Request - Repository

```
@Repository
public class TVShowRepository {
    @Autowired
    private JdbcTemplate template;

    public SqlRowSet get(final Integer tvId) {
        return template.queryForRowSet(
            “select * from tv_shows where prog_id = ?”, tvId);
    }
}
```

Handling a Request - Data Transfer Object

```
public class TVShow {
    private String tvId;
    private String name;
    ...

    // getter and setters
    public String getTVId() { return tvId; }
    public void setTVId(String id) { tvId = id; }
    ...

    public static TVShow populate(SqlRowSet rs) {
        final TVShow tv = new TVShow();
        tv.setTVId(rs.getString("prog_id"));
        tv.setName(rs.getString("name"));
        ...
        return (tv);
    }
}
```