Day 22

Aggregation

- Perform calculations on a set of records and returns a single value
  - Eg. Total number of TV programs releases this month
  - Eg. Total orders for this month
- Aggregation functions includes
  - Sum, count, average, min, max, distinct, standard deviation, etc
- https://dev.mysql.com/doc/refman/8.0/en/group-by-functions.html

`distinct` Keyword

```
select distinct lang
  from tv_shows

select
  distinct lang, rating
  from tv_shows
```

- Returns distinct values from a column
  - Remove the duplicates
- Used on columns with discrete values
  - Discrete value - **`lang`**, **`rating`**
  - Continuous value - **`user_ratings`**

`count` Keyword

```
select count(*)
  from tv_shows
  where lang like 'English'
```

Count the number of English language TV programs

```
select count(distinct name)
  from tv_shows
  where lang like 'English'
```

Count the number of English language TV programs with no duplicates

Arithmetic Aggregation

```
select avg(user_rating)
  from tv_shows
  where lang like 'English'
```

Average ratings for English language TV shows

```
select sum(length(image))
  from tv_shows
  where lang like 'English'
```

Image size for English language TV shows

```
select sum(user_rating) / count(*)
  from tv_shows
  where lang like 'English'
```

Average ratings for English language TV shows without using the **`avg()`** function

Aggregating by Category

```
select rating, count(rating)
  from tv_shows
  group by rating;
```

Aggregate the number of TV programs for each rating category

```
 select rating, count(rating)
   from tv_shows
   group by rating
   order by count(rating) desc;
```

Order the rating in descending order. Default is ascending

```
select rating, count(rating)
  from tv_shows
  where lang like 'English'
  group by rating
  order by count(rating) desc;
```

Aggregation with predicate

Aggregating by Category

```
select rating, count(rating)
  from tv_shows
  group by rating
  having count(rating) > 100;
```

Only select the rating where the count is greater than 100

```
select rating, count(rating)
  from tv_shows
  where count(rating) > 100
  group by rating;
```

Cannot use aggregate functions on `where` clause

`where` and `having`

`where`	`having`
Used as a filter to select individual records by the query. If record does not satisfy the `where` clause, it will not be selected by the query	Used as a filter to select results from aggregation. `having` is applied after the records have been selected by a query
Applied before `group by`	Applied after `group by`
Cannot contain aggregation operators	May contain aggregation operators

where and **having**

```
select rating, count(rating)
  from tv_shows
  where lang like 'English'
  group by rating
  having count(rating) > 100
  order by rating desc;
```

Select from the table that satisfy there `where` clause

Only group the result of the query that satisfy the `having` clause

Order the results according to the following

as Keyword

- Used to alias a column name
  - Eg. give use friendly names to computed columns
  - Makes reading the result easier

```
select count(*)
   from tv_shows
   where lang like 'English'
```

```
+---------+
| count(*)|
+---------+
|   42    |
+---------+
```

rowSet.getInt("count(*)")

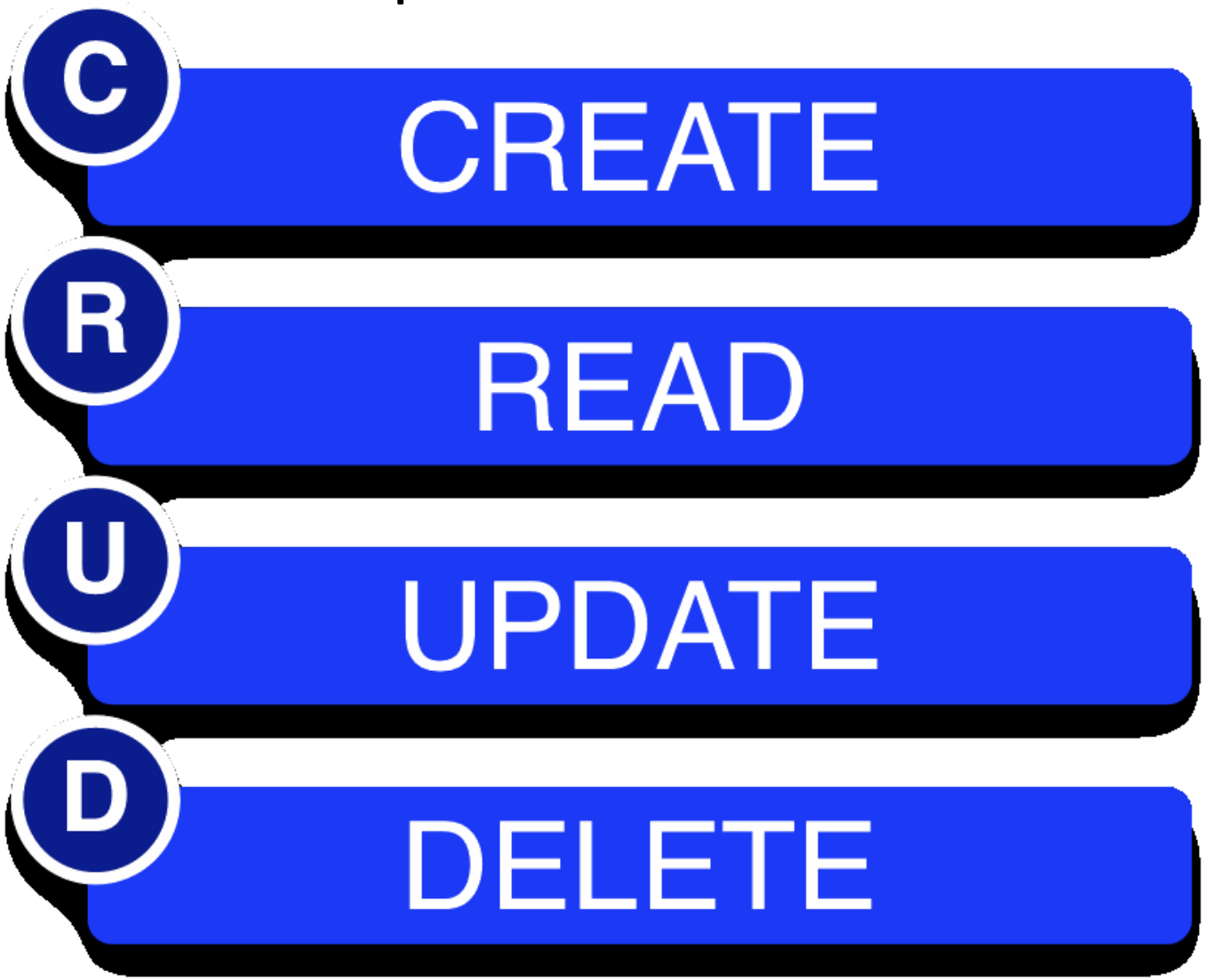
Alias for this column

```
select count(*) as show_count
   from tv_shows
   where lang like 'English'
```

```
+------------+
| show_count |
+------------+
|    42      |
+------------+
```

rowSet.getInt("show_count")

Major Database Operations



Create - INSERT

Table name

List of columns names in the table.

```
insert into tv_shows
    ( prog_id, name, lang, rating, release_date)
values
  ( 1, "Arrow", "English", "PG", "2012-10-10");
```

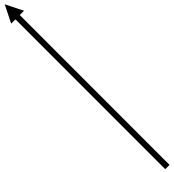
The order of the values listed must be in the same order as the column names in the column list

The data type must also match the data type of the column

Can skip columns if the columns are ‘nullable’

Create - INSERT

```
insert into tv_shows values
  ( 1, "Arrow", "English", null,
    "PG", 6, "2012-10-10", null);
```



The list of values must match the order of the column names in the table

Use null as placeholder for missing column values

Update - Update

Table name

Indicate the fields to be updated using `set`

LHS field name, RHS the value to update the field

```
update tv_shows set
    name = "Breaking Bad",
    release_date = "2008-01-20"
    where prog_id = 10001;
```

Condition of the update

Primary keys should not be updated

Delete - DELETE

Delete the record where `prog_id` is equal to 10001

Use a `select` to verify the where clause first before deleting if the condition is complex

```
delete from tv_shows where prog_id = 10001;
```

```
delete from personality;
```

Updates

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

    pubic boolean add(final TVShow tv) {

        int added = template.update(
            "insert into tv_shows(prog_id, name, ....) values (?, ?, ...)",
            tv.getTVId(), tv.getName(), ...);

        return added > 0;
    }
}
```

Update to execute SQL statements that updates the database

Any prepared update statements eg. insert, update or delete

Returns the number of rows affected by `update()`

Batch Updates

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

    pubic int[] add(final List<TVShow> shows) {
        List<Object[]> params = shows.stream()
                .map(tv -> new Object[]{ tv.getTVId(), tv.getName(), ... })
                .collect(Collectors.toList());

        int added[] = template.updateBatch(
            "insert into tv_shows(prog_id, name, ....) values (?, ?, ...)",
            params);

        return added;
    }
}
```

Create List of Object array that matches the order of the prepared statement

List of params for the prepared statement

Each element contains the number of row affected by each update

`added.length == params.size()`

Naming Resources

- Resources should be rooted under a common namespace
  - Eg `/api`, `/api/v1`
- Use plural for resource collections
  - Eg `/api/tv_shows`
- Use singular for a single resource
  - Eg `/api/tv_show/1`

Operations on Resources

- `GET` – get the resource
  - Eg. `GET /api/tv_show/10005`
  - Eg. `GET /api/tv_shows`
- `POST` – creates a new resource
  - Eg. `POST /api/tv_show/20007`
- `PUT` – updates a resource
  - Eg. `PUT /api/tv_show/10005`
- `DELETE` – removes a resource
  - Eg. `DELETE /api/tv_show/20007`

Resource Collections

- `GET` – get the resource
  - Eg. `GET /api/tv_shows`
  - Eg. `GET /api/tv_shows/nc16`
- Collections should return hyperlinks to resources rather than the resources themselves

```
{
  rating: 'NC16',
  tv_shows: [
    '/api/tv_show/10005',
    '/api/tv_show/20007',
    ...
  ]
}
```

Links to resources

Handling Error

- `error.html` is the default error page
  - Have to populate the model before displaying the
  - Only return HTML payload
- Spring provides a general purpose way of handling error
  - Throw exception, error handlers can target specific exception
  - Custom exception can carry any data to the error handlers
  - Centralize all error handling to a few classes
- Error response can be
  - `String` or `ModelAndView` for HTML
  - `ResponseEntity<T>` for more general response eg. JSON

Example - Custom 404

Annotate the exception class with the status code. This is optional because the status code can be set when the exception his processed

```
@ResponseStatus(HttpStatus.NOT_FOUND)
public class RecordNotFoundException<T> extends RuntimeException {
    private final T primaryKey;
    private final String tableName;
    public RecordNotFoundException(String table, T pk) {
        super(String.format("Cannot find record in %s table with primary key %s"
            , table, pk);
        primaryKey = pk;
        tableName = table;
    }

    public T getPrimaryKey() { return primaryKey; }
    ...
}
```

Exception can delivery any data

Subclass RuntimeException

Example - Exception Handler

Annotate class

```
@RestControllerAdvise
public class ErrorController {

    @ExceptionHandler({ RecordNotFoundException.class })
    public ModelAndView handleRecordNotFound(
            HttpServletRequest req, RecordNotFoundException ex) {

        final ModelAndView mav = new ModelAndView("record_not_found.html");
        mav.addObject("tableName", ex.getTableName());
        mav.addObject("primaryKey", ex.getPrimaryKey());
        mav.setStatus(HttpStatus.NOT_FOUND);
        return mav;
    }
}
```

One or more exception to be handled by this error handler

List of bean that can the handler can access. The most important is the actual exception

Flexible way of associating a model with a view. Instantiate `ModelAndView` with the view's name

Return the dynamically constructed view

Populate the model like `model.addAttribute()`

`HttpServletRequest` provides information on the request like query string, headers, etc

Optionally set status code