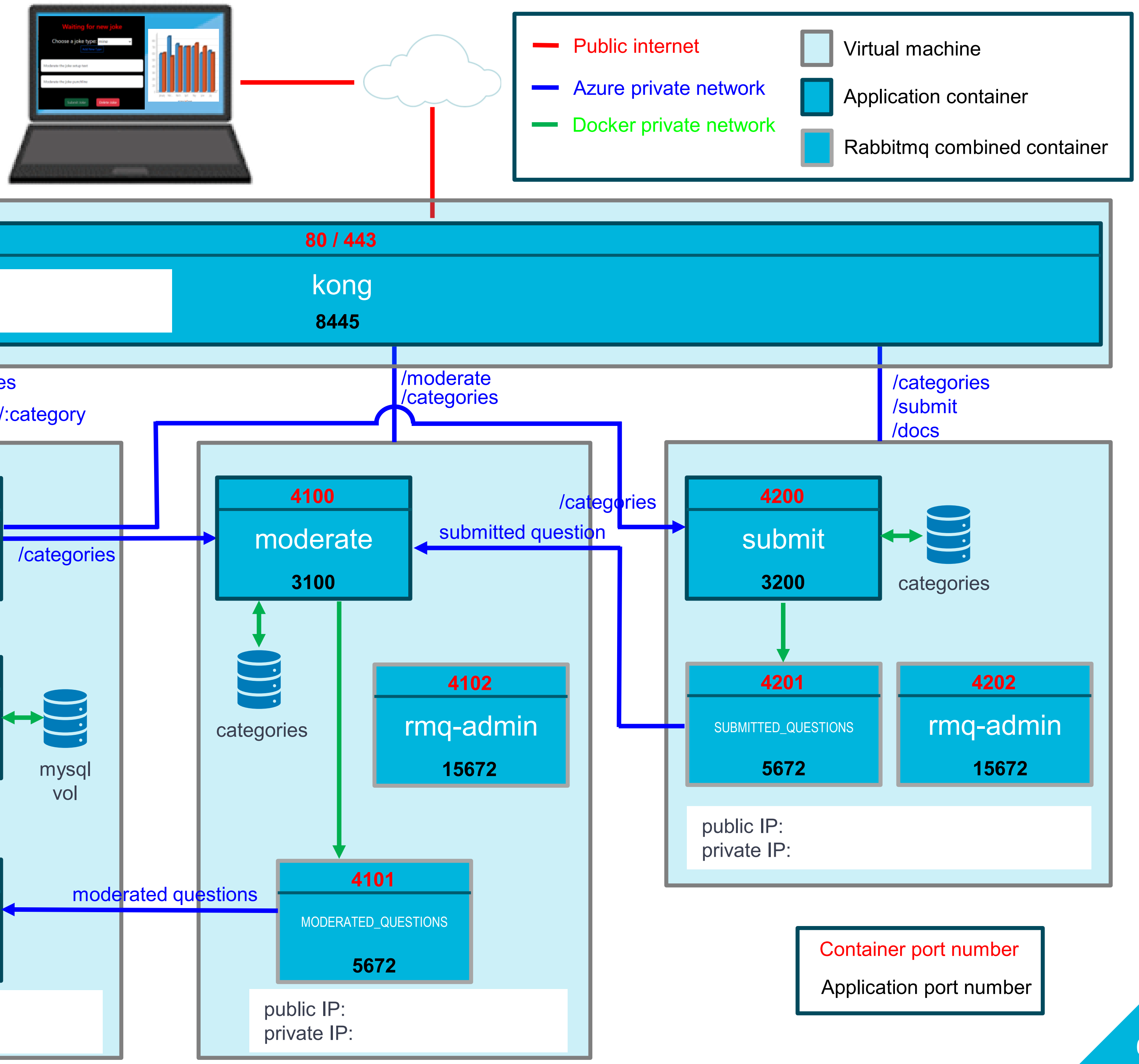
Adding a database to the stack

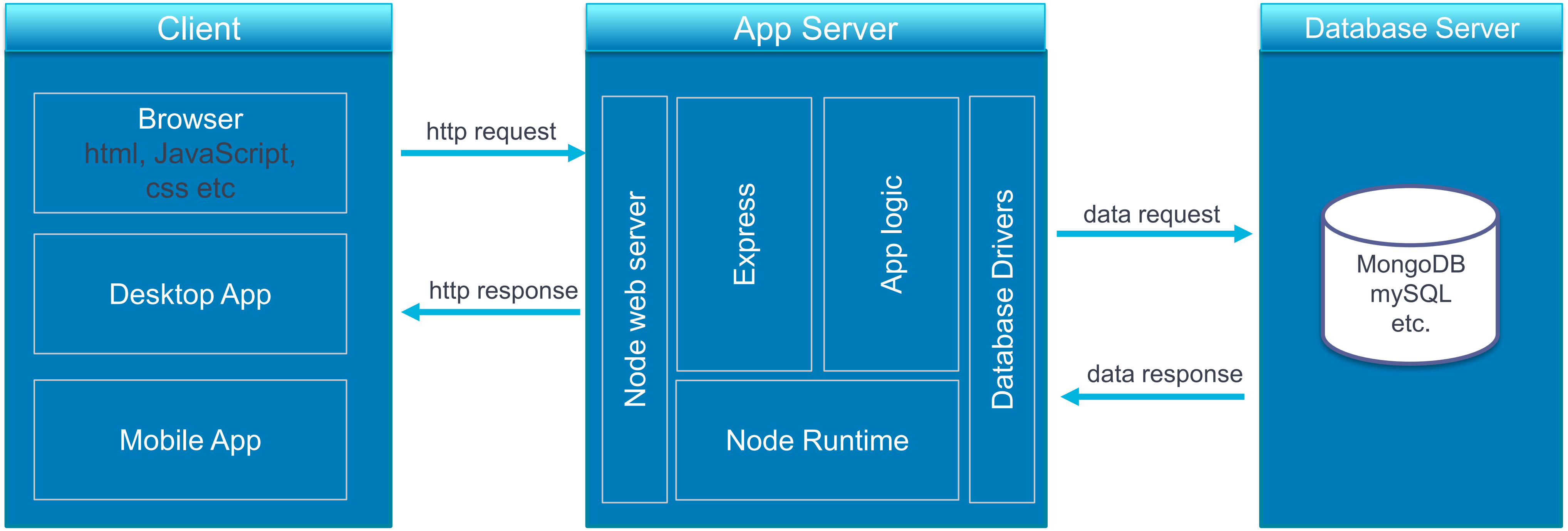
Learning objectives

- Upon successfully completing this section, you will be able to:
  1. Explain the ETL process
  2. Explain cardinality in an entity relationship diagram
  3. Explain and create table relationships using primary and foreign keys
  4. Explain and create referential integrity rules
  5. Interpret and write basic SQL
  6. Explain SQL injection and how you can prevent it
  7. Explain and use the different scopes of environment variables
  8. Explain the benefits of using environment variables in maintainability and application configuration
  9. Explain the benefits of database connection pooling in enterprise systems
  10. Explain and implement database connection pooling which manages multiple concurrent connections
  11. Create an application which adheres to the principle of separation of concerns

Adding a Database



Architecture



- So we need Node.js to run any back-end code in JavaScript and provide the static web server
- We need Express.js to simplify creating the server and to process requests and return responses
- We need to write some logic – i.e. our app, data access and processing
- We need an appropriate database driver / SDK to enable our code to access the database
- We need a database management system / database server

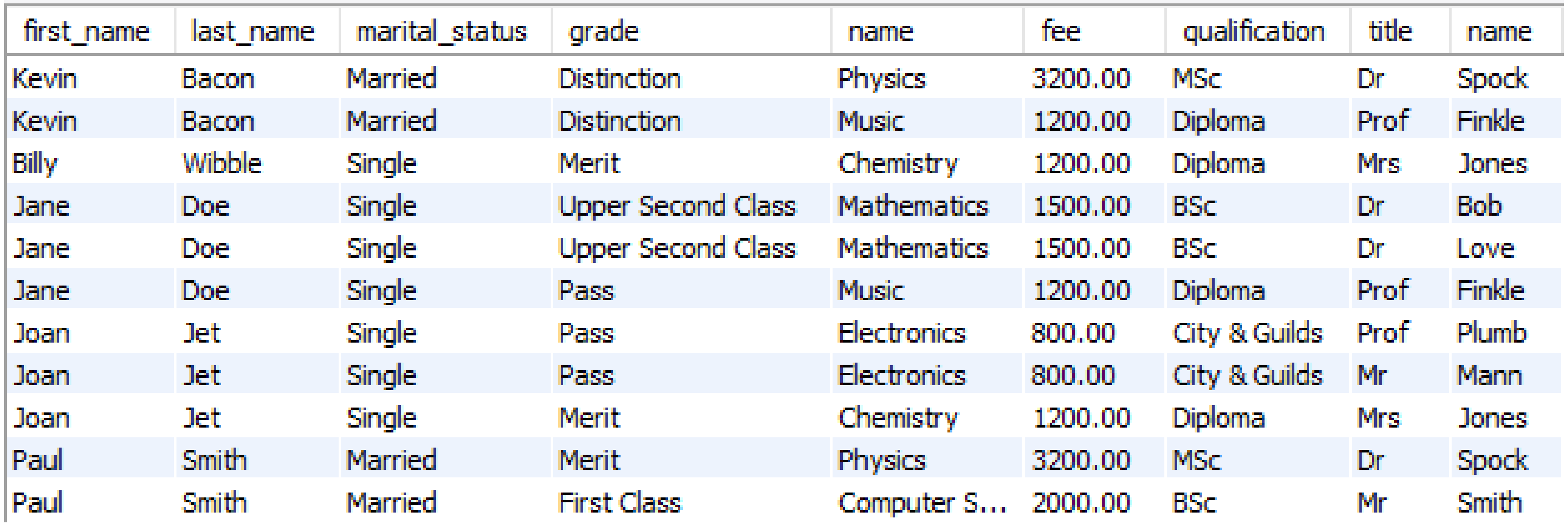
- It’s time to add a database to complete the 3-tier architectural stack
- There are various database types, mainly relational and various no SQL
- We will look at relational first using MySQL server, now owned by Oracle
  - You should install mySQL server and Workbench from https://dev.mysql.com/downloads/mysql/8.0.html
- This is not a database course, you did enough in the first year to cover this module, so I’ll stick to basics
- MySQL is a very popular relational database system, RDBMS, but since Oracle took it over, it has been forked and some prefer to use the fork, MariaDB. They are basically the same, but we will use MySQL
- A relational database uses multiple tables which are related in some way. A process of data normalisation is used to remove repetition from the data and determine relationships
  - If you can't remember how to normalise from the first year, just look it up as revision
  - We will not create complex data models or require in-depth normalisation
- We'll do this in two parts
  - Quick and dirty to get the database working and integrated using a single simple table
  - A more engineered approach

demos/data/basic/app.js (create db and table manually)

demos/data/mysql2_basic_promise/app.js

Data Engineering

- Software Engineers work closely with Data Engineers so need a good grasp of data operations
- To summarise a software developer should be able to analyse, standardise, **E**xtract & clean, **T**ransform, and **L**oad (ETL) data into data structures they or a data engineer have modelled to enable others to use and analyse - i.e. create related normalised tables and write the data to the tables if using RDBMS
- Various tools are used in this process, or you may write code for continuous ETL into a database
- e.g., one system creates a log of all the changes made to its database each day. In the evening, a batch process job may be run to transfer that data to an analytical system to be added to data from other systems for analysis. That data may be transferred as comma separated (or other) values
- A data engineer / software developer will need to clean the data
  - scan for missing column values, blank records, ensure compliance to data standards etc
- They will **extract** the data from the file in in whatever format it is in
- They will **transform** the data - e.g. set all names to upper or lower case, make sure date formats are correct etc, align to organisational data standards, maybe separate data based on table destination
- Then **load** the data into a table or a set of related tables
- The ETL process is a very popular process in data processing systems

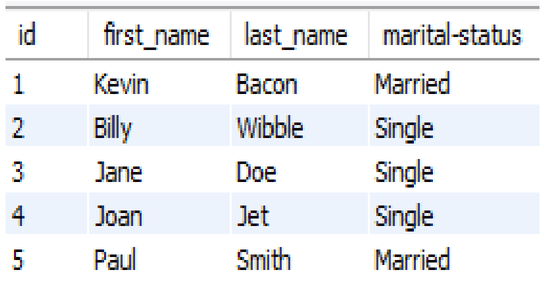


first_name	last_name	marital_status	grade	name	fee	qualification	title	name
Kevin	Bacon	Married	Distinction	Physics	3200.00	MSc	Dr	Spock
Kevin	Bacon	Married	Distinction	Music	1200.00	Diploma	Prof	Finkle
Billy	Wibble	Single	Merit	Chemistry	1200.00	Diploma	Mrs	Jones
Jane	Doe	Single	Upper Second Class	Mathematics	1500.00	BSc	Dr	Bob
Jane	Doe	Single	Upper Second Class	Mathematics	1500.00	BSc	Dr	Love
Jane	Doe	Single	Pass	Music	1200.00	Diploma	Prof	Finkle
Joan	Jet	Single	Pass	Electronics	800.00	City & Guilds	Prof	Plumb
Joan	Jet	Single	Pass	Electronics	800.00	City & Guilds	Mr	Mann
Joan	Jet	Single	Merit	Chemistry	1200.00	Diploma	Mrs	Jones
Paul	Smith	Married	Merit	Physics	3200.00	MSc	Dr	Spock
Paul	Smith	Married	First Class	Computer S...	2000.00	BSc	Mr	Smith

- There are duplicates
- If you wanted to find out Joan Jet's grade in Electronics, how would you do it?
- Searching will return two results - there could be two Joan Jets - there is no unique ID
- If it's the same person and she had her grade changed on appeal, if not all records updated then data is inconsistent
- There are two columns called **name**
- We need some uniqueness and to avoid checking many columns, so break the table up into tables

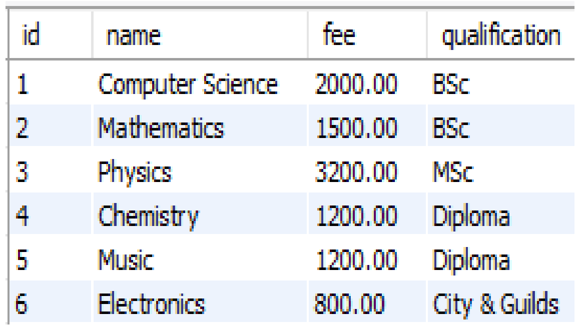
- A student can be on more than one course, but for simplicity, a tutor can only teach one course, but a course can have more than one tutor. I just made this up, but this information would be elicited from the business by a business analyst working in conjunction with an engineer
- Split into entities of related information using data normalisation, e.g. student, course and tutor. Each is given a unique ID to search on, this is the primary key. There are no duplicates
- The tutor table enables us to find the course they teach using the course primary key, e.g. staff number
- Grade is a many-to-many relationship as many students can be on a course and a course can have many students but a grade is unique to one student on one course. A single primary key on grade would give us nothing useful but a primary key made of two other primary keys to form a composite primary key is useful as it relates the grade to the student **and** the course. We deal with this scenario with a linking or joining table with both key values. e.g. What grade did Kevin Bacon get for Physics?
- There's repetition in grade, student and course - so marital-status, qualification and grade could have their own tables and be replaced by foreign keys. That would require more joins which may impact performance so it's a trade-off
- The 4 tables, functionally dependent on a key that will work are:

student



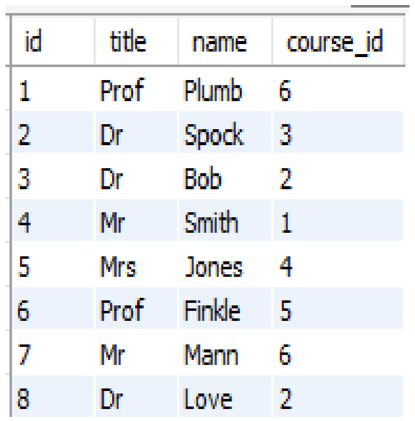
id	first_name	last_name	marital-status
1	Kevin	Bacon	Married
2	Billy	Wibble	Single
3	Jane	Doe	Single
4	Joan	Jet	Single
5	Paul	Smith	Married

course



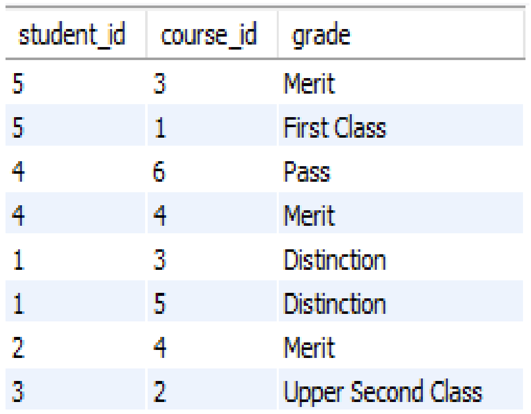
id	name	fee	qualification
1	Computer Science	2000.00	BSc
2	Mathematics	1500.00	BSc
3	Physics	3200.00	MSc
4	Chemistry	1200.00	Diploma
5	Music	1200.00	Diploma
6	Electronics	800.00	City & Guilds

tutor



id	title	name	course_id
1	Prof	Plumb	6
2	Dr	Spock	3
3	Dr	Bob	2
4	Mr	Smith	1
5	Mrs	Jones	4
6	Prof	Finkle	5
7	Mr	Mann	6
8	Dr	Love	2

grade

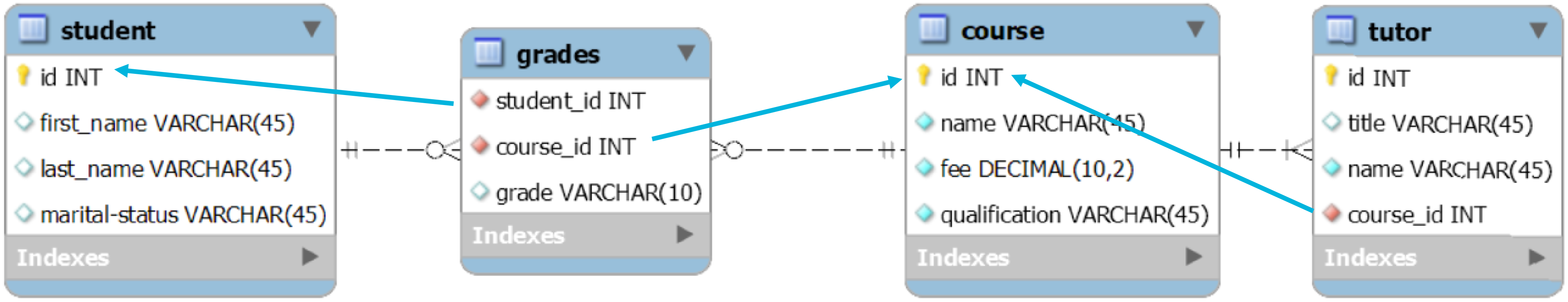


student_id	course_id	grade
5	3	Merit
5	1	First Class
4	6	Pass
4	4	Merit
1	3	Distinction
1	5	Distinction
2	4	Merit
3	2	Upper Second Class

Data modelling

- You would likely be designing a new product and don't yet have any data. You work with the business analysts to determine what data you need.
- The design would be used by data engineers and software engineers to create the solution
- Modelling it is typically done using an entity relationship diagram ERD in a modelling tool such as Erwin
- MySql workbench has a basic modelling tool built-in so I used that to create the following entity relationship diagram (ERD) diagram but the tables we will use are easily normalised and modelled without this but it's handy for documentation
- Workbench can take an existing database and create an ERD from it
- Workbench enables you to draw the diagram and create all the SQL to create the tables from the diagram
- There are more professional tools for enterprise designs with many more features

The notation in this example is referred to as "crow's foot" notation



Left to right: each student can have zero or more grades

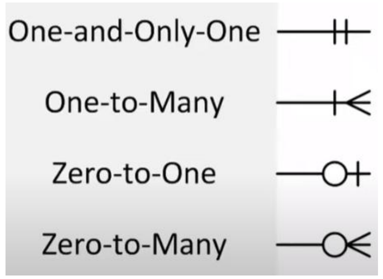
Right to left: a grade can be associated to 1 student.

Left to right: A grade can be associated to only one course

Right to left: a course can have 0 or more students.

Left to right: A course can have only 1 tutor

Right to left: a tutor can teach on only one course (in this example)



Decent tools show the relationship between primary and foreign key so I've added the blue arrows
We may want a direct relationship between student and tutor. Is there a relationship between student and tutor beyond the course? Personal tutor? This is all part of requirements gathering

demos\data\students_db
reverse: sql_store DB

- With the database designed we can create it by forward engineering an ERD or manually
- Relational databases are accessed and operated using the Structured Query Language (SQL)
- With SQL we can create a database, tables, add data, modify data, request data etc
- You can use the workbench GUI but it still shows you the SQL you could have used

- We will only be using a subset of SQL so I'm not going to go through it all - there are many references
  - `SELECT * FROM student_db.course;` To avoid having to specify the database every time double click schema or specify it: **use student_db;**
  - `SELECT name, qualification FROM course;`

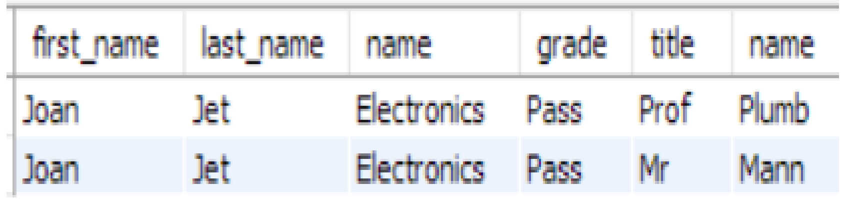
  -
    ```
    INSERT INTO student (first_name, last_name, marital_status)
    VALUES ('Tony', 'Nicol', 'Married');
    ```
  - `DELETE FROM student WHERE id=7;`

  - If the data you need is in more than one table, you need to join them to create a resultant set. e.g., we want to list all tutors and the courses they teach. It is customary to put SQL keywords in capitals but not required:
    ```
    SELECT tutor.title, tutor.name, course.name
    FROM tutor
    JOIN course on course.id = tutor.course_id;
    ```

Demo some queries

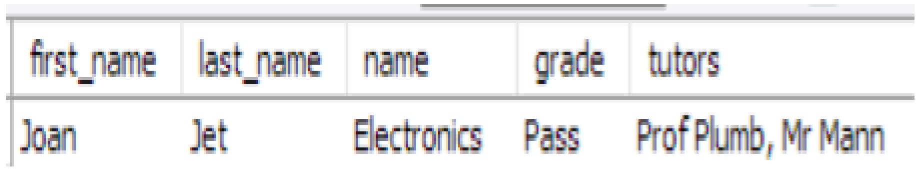
- Join all tables to find what courses and grades Joan Jet is on and the tutor names
- This will create multiple rows if there are multiple tutors. If you don't want that you can use concat

```sql
select first_name, last_name, course.name, grades.grade, tutor.title, tutor.name
from student
  inner join grades on student.id = student_id
  inner join course on course.id = grades.course_id
  inner join tutor on course.id = tutor.course_id
where student.last_name = "Jet" and course.name = "Electronics";
```



first_name	last_name	name	grade	title	name
Joan	Jet	Electronics	Pass	Prof	Plumb
Joan	Jet	Electronics	Pass	Mr	Mann

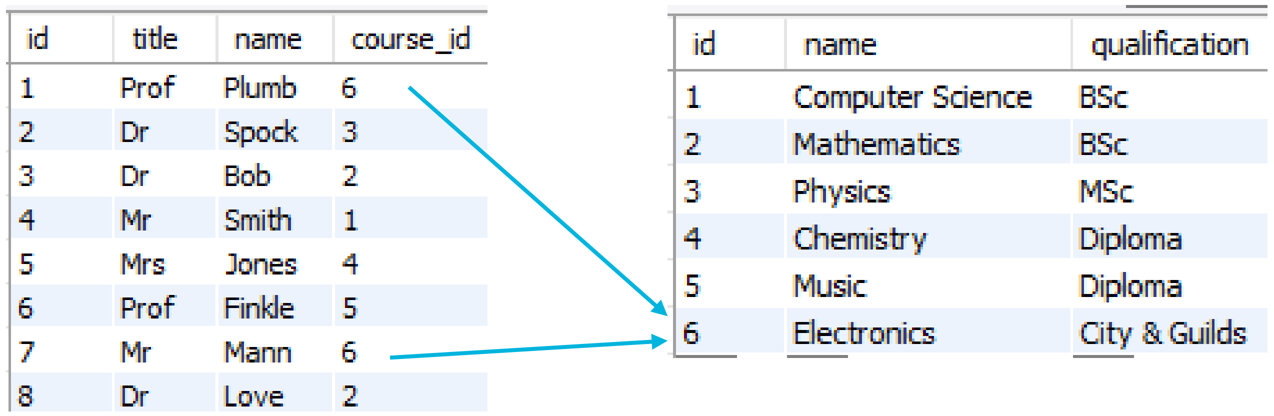
```sql
SELECT
    student.first_name,
    student.last_name,
    course.name,
    grades.grade,
    GROUP_CONCAT(CONCAT(tutor.title, ' ', tutor.name)SEPARATOR ', ') AS tutors
FROM student
INNER JOIN grades ON student.id = grades.student_id
INNER JOIN course ON course.id = grades.course_id
INNER JOIN tutor ON course.id = tutor.course_id
WHERE student.last_name = 'Jet' AND course.name = 'Electronics'
GROUP BY
    student.first_name,
    student.last_name,
    course.name,
    grades.grade;
```



first_name	last_name	name	grade	tutors
Joan	Jet	Electronics	Pass	Prof Plumb, Mr Mann

Referential integrity

- If you think about the structure of related tables, say we have 1000 employees in one building, site(1) and a similar number in other buildings around the country, site(n). You would likely have an employee table and a site table at least where each employee has a foreign key value in the site table
- Say we want to list all employees in one site - we need to join the tables
- Now say someone accidentally deletes site 3 from the site table, this is a parent table. None of the entries in the child table, employees, has a site. Their parent has been deleted - they are orphans - like losing a pointer
- So, you think, ok, I'll just add the site back in. But if you do, and you are using auto-generated keys, the primary key will be a different value to the foreign key, so you gain nothing
- This needs some significant manual intervention
- To avoid this, we can protect the relationship by setting referential integrity rules
- Using the student database. If we dropped the course table, no member of staff would know what course they were teaching, and students couldn't be matched with a course for their tutor or grades
- Even deleting one row, e.g., electronics, would affect all the people involved with that course
- Accidents do happen. Let's protect course against this

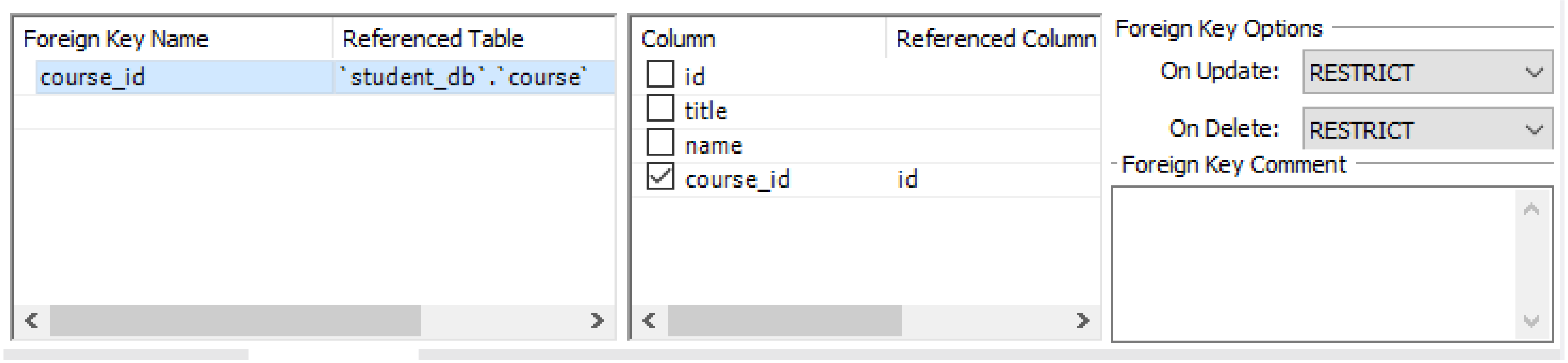


id	title	name	course_id
1	Prof	Plumb	6
2	Dr	Spock	3
3	Dr	Bob	2
4	Mr	Smith	1
5	Mrs	Jones	4
6	Prof	Finkle	5
7	Mr	Mann	6
8	Dr	Love	2

id	name	qualification
1	Computer Science	BSc
2	Mathematics	BSc
3	Physics	MSc
4	Chemistry	Diploma
5	Music	Diploma
6	Electronics	City & Guilds

- Deleting electronics then re-adding would require all course id of value 6 to be changed to 7
- Imagine a database with millions of people, with many relationships - I've seen systems with over 100 tables joined. Finding the orphans and fixing them can be very tricky.
- Alternatively, you really do want to delete Electronics as you will no longer deliver it as a course. So, you delete it but all the references to it, in this case just tutors and grades, will be stuck there as orphans. If the course is removed, all references to it need to be removed too. That's very tricky in a large complex database

- To provide referential integrity between course and tutor, you can add a constraint onto the keys. On delete, you can **cascade**, **restrict**, **set to null** or **do nothing**.
- Cascade means it will automatically delete all the children of a deleted parent. E.g., deleting a row in the course table would delete all tutor entries for that course so leaves no orphans
- Restrict means, if there are records in the child table with reference to the parent you are trying to delete, don't allow the delete and throw an error
- There are the same options associated with update. If you update a parent record, e.g., change the course primary key from 6 to 7 for some reason, the cascade constraint will automatically update all child records with key value 6 to 7. If it is set to restricted, then trying to update a parent record will raise an error
- Alter the tutor (child) table | foreign keys | select key. Referenced column is id. By default, these are restricted



Foreign Key Name	Referenced Table
course_id	\`student_db\`.\`course\`

Column	Referenced Column
☐ id	
☐ title	
☐ name	
✓ course_id	id

Foreign Key Options

On Update: RESTRICT

On Delete: RESTRICT

Foreign Key Comment

Separation of Concerns

SQL Injection

- So the database has various protections but what about our code?
- SQL injection is where typically an application requires data from a user to say, look something up
- For example, an API where we want to get details on someone with a last name of martin
  - `localhost:3000/name/martin` The name would typically be read in the web UI

    /demos/mysql2_basic_promise_sql_injection
  - But what if we passed ``localhost:3000/name/`' OR 1=1;  -- ` ``
- If we can determine column names we can modify content - e.g. give ourselves a 1st class degree
  - ``localhost:3000/name/`'; INSERT INTO names (first_name,last_name,age) VALUES ('tony','nicol',25);--` ``
- How do we protect against this?
  - Database configuration - `multipleStatements: false (default in mySQL2)`
  - Parameterised queries
  - Thorough code checking of what's passes in

Parameterised queries

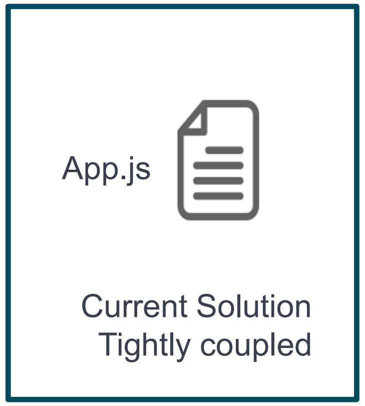
- The previous example combines code and data; we need to separate these
- We need to make sure user input is treated only as data and is not executable SQL
- Data passed in for query is replaced with a placeholder, '?' and the data is added to an array
- The query with ? placeholders is passed as arg1 and data array as arg1 - the function safely combines
- Say we pass in 3 params in the path or in a POST body
  - `localhost:3000/student/billy/wibble/25`
  - `const params = [req.params.fName, req.params.lName, req.params.age]`
  - `const sql = "SELECT * FROM names WHERE first_name=? AND last_name=? AND age =?"`
  - `const [result] = await connection.execute(sql, params)`

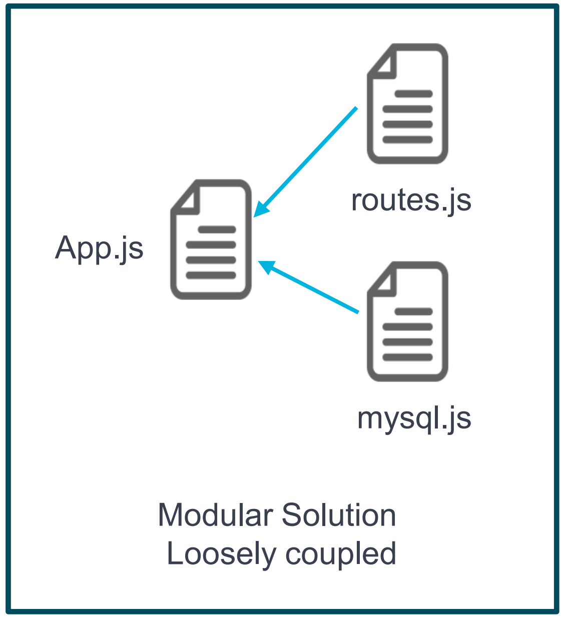
- Note: **.query** works ok but **.execute** is preferred for security and query performance with ? params

/demos/mysql2_basic_promise_sql_injection

Separation of concerns

- Means designing a solution such that different parts of the code or architecture handle **distinct responsibilities**, without overlapping or interfering with each other
  - Each microservice/module/class/function has a single, well-defined purpose
  - Changes in one concern don't impact the others
  - Code is more maintainable - easier to test and extend





Environment variables

- Before we get into modular database-application design, a quick sidetrack on environment variables
- Environment variables are variables held by the operating system so can be made accessible to applications. For example, the system scope env vars such as PATH in Windows, holds the paths to applications so you can access applications from any directory
- We can set env vars that are scoped to a process - i.e. a running app. These are not visible to other processes
- Our process (e.g. node app) can access our process scoped env vars **and** the inherited shell and system env vars
- We use environment variables to keep some of the sensitive or common settings out of our source files
- We can also use them to configure our application for different circumstances and environments at runtime
- They act like parameters to an app for flexibility - not sent to the app but available for the app to read
  - e.g. start server on a different port without changing code
- We can set env vars manually (don't leave space either side of =
  - Windows power shell: `$env:PORT=4000` View with `$env:PORT`
  - Windows command shell: `set PORT=4000` view with `set PORT`
  - Linux: `export PORT=4000` View with `echo $PORT`
  - `const port = process.env.PORT` will read PORT from the environment; use it to start server on 4000

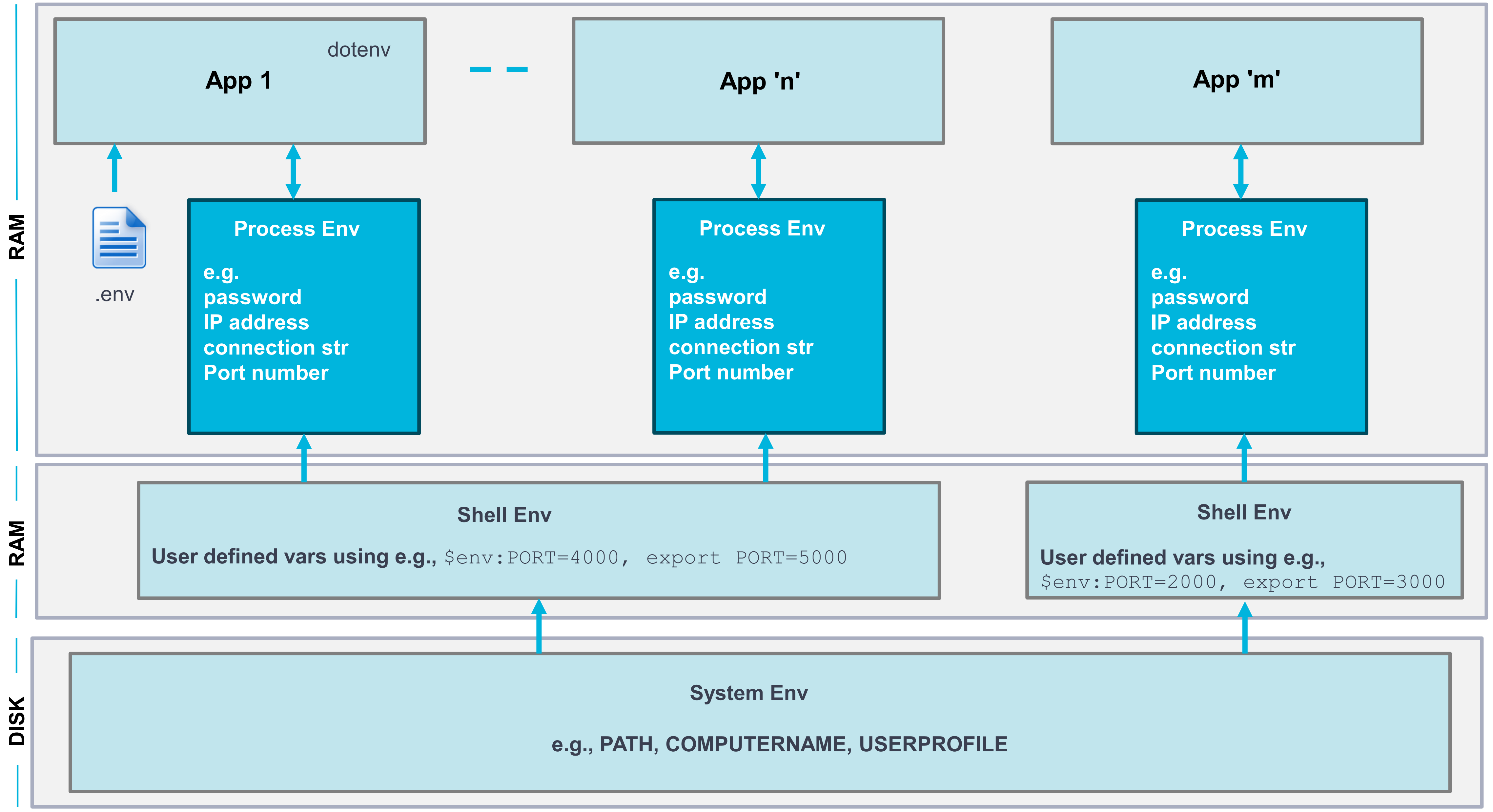
- Rather than set the environment variables manually, we can use an npm package **dotenv**
- Write your environment variables in a file - default is .env. Include the npm package ‘dotenv’. When the dotenv config function is called, it will read the variables from the file and write them into the process environment

  ```
  require('dotenv').config()
  ```
- Create a file called .env in the root dir of your project.
- If you want to put the file somewhere else or you are having trouble opening it, use the following to make it explicit:

  ```
  require('dotenv').config({ path: path.resolve(__dirname, '.env') })
  ```
- Using this approach you can call the file something else if you want to.
  - Note: path needs to be "required" but doesn't need npm as it is already included in modern versions of node
- To see all your env vars:
  - Windows powershell: `Get-ChildItem Env:` or `Get-ChildItem Env: | Sort-Object Name` to sort them on name
  - Windows command shell: `set`
  - Linux: `env` or `env | sort`
- Using .env to hold passwords is a little more secure than adding them in the code. If you are going to push to github, don't commit your .env file. To prevent that, add .env to a .gitignore file (add ./node_modules too)
- If you already have an env var declared in your env, dotenv won't overwrite it - i.e. a shell var trumps a .env var

/demos/data/dotenv

Basic Env Var Architecture



Database connections

- When a query from the app is made on the database, typically in the simplest non-busy scenario
    - a connection object is created
    - the app authenticates using the connection string
    - the query is completed
    - the app closes the connection
    - improves security by minimising session hijacking or unauthorised reuse in very secure environments - limited time for hacker
    - may use per-user credentials for isolation and auditing

- Alternatively, a connection object can be created and kept open until the application closes
    - only take the object creation and authentication hit once so improves performance
    - slightly less secure but fine for all but the most sensitive data scenarios
    - cannot run concurrent queries needed in a multi-user environment - e.g. concurrent http requests
    - or even multiple queries from the same user to say restore a user profile when they login to the web app

Database Connection Pooling

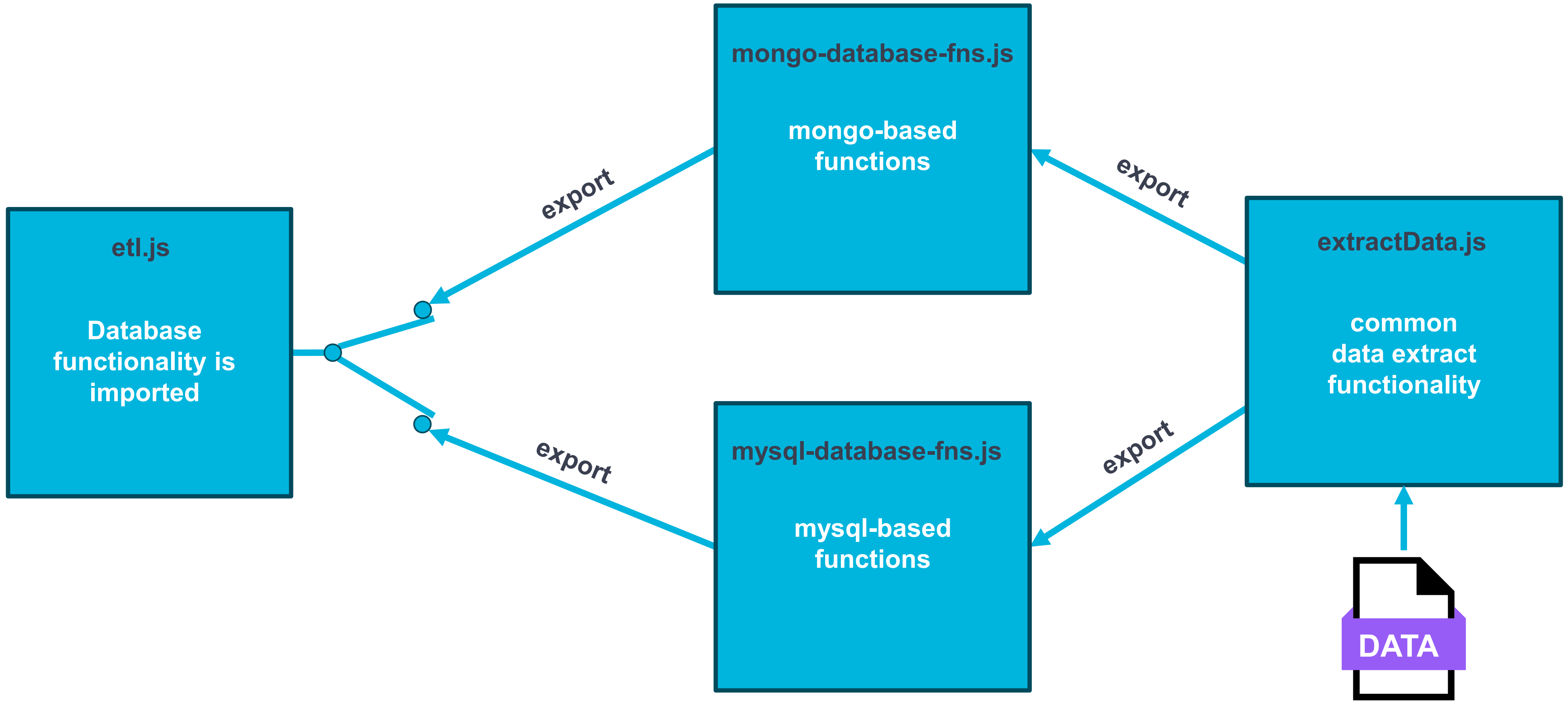
- Connection pooling is a technique used to manage and reuse database connections efficiently
  - Especially in high-performance or multi-user applications
  - Avoids the time overhead of opening and closing a new connection for every database request
  - Creates a pool of reusable connections when the application starts and authenticates them all
  - Hands out an available connection from the pool when a query is made & returns it on query completion
  - Keeps connections alive and ready for reuse, reducing overhead of creating and authenticating
  - Supports concurrent users and concurrent queries from the same user
  - Can prevent connection leaks by using auto lifecycle management - pool.query() gets, uses then releases connection
- Can set limits. e.g., for mySQL2
  - connectionLimit: limit concurrent connections to prevent resource exhaustion such as CPU, RAM, I/O. Default=10
  - waitForConnections: should pool exhaustion cause connection requests to be queued. Default=true
  - queueLimit: if queuing, how big do we allow the queue to grow? Default=0 which means unlimited

/demos/data/mysql2_promise_modular

Data engineering use-case

- I have a pdf document with trivia questions and answers. It's formatted with headers, footers, page numbers etc. I want to extract the data, transform it into a format I need to be able to load it into a database which I have modelled
- Step1: I exported into a word document so I could edit it
- Step2: I needed to clean the data - replace smart quotes, en & em dashes, other non standard characters into standard characters. Remove headers, footers, section breaks etc. This is automated using word's search and replace feature
- Step3: Data were exported into delimited text: tab to separates a question from an answer; newline to separate each question-and-answer pair. Database special characters need to be dealt with too - e.g., " and ' are used in databases to separate data but the date may contain " or '. So, all " need to be converted to either "" or \" and ' to '' or \'. There are others. You could use backtick to avoid the change
- Step 4: **Extract**. Data is loaded from storage, e.g. a text file which is typically coma separated values (csv) or delineated in other ways (tab and eol in my case), maybe the format is json. Whatever, it needs to be extracted to some form of staging area so the original is not modified
- Step 5: **Transform**. Any more formatting and cleaning to get the data into a form that maps to our data table design
- Step 6: **Load**. Data is loaded into one or more tables in a database
- In this simple example, the database has only one table with three columns: id, question and answer
- The ETL is written in modular JavaScript. It loads data from a file, separates question and answer, creates a table then loads all the data from the file into the table.
- A use case could be: datafile lands. app.js triggered to do etl. Data file archived. Data available for API request

- The modular structure enables the database layer to be changed without changing the app



/demos/data/etl/app.js