Databases and Cloud Concepts Notes

paraphrased by Tyler Wright

*An important note, these notes are absolutely **NOT** guaranteed to be correct, representative of the course, or rigorous. Any result of this is not the author's fault.*

Contents

1 The Internet

The internet is a world-wide computer network, connecting computing devices also known as hosts or end systems. These connections can take many forms, such as cables and radio waves. Intermediate switching devices inbetween hosts are known as routers.

1.1 Clients and Servers

A program or machine that responds to requests from others is called a server. A program or machine that sends requests to a server is a client.

1.2 Internet Layers

There are four internet layers:

Layer	Common Protocol	Description
Application	HTTP	Web browsers making requests and parsing responses
Transport	TCP	Breaks requests down into numbered packets and can reassemble messages
Network	IP	Attaches addresses to packets and groups packets based on their incoming addresses
Physical		Sends bits to the local router and assembles bits into packets

1.3 Protocols

Protocols are an agreement on how to communicate.

1.3.1 HTTP - HyperText Transfer Protocol

There are four main operations that can be carried out on HTTP resources:

Operation	Performed by...
Creation	`HTTP POST`
Reading	`HTTP GET`
Updating	`HTTP PUT`
Deletion	`HTTP DELETE`

Requests are formed by an operation as well as a `host` and `content-type` parameter to describe the format of information.

1.3.2 URL - Uniform Resource Locator

Each URL is formed by a scheme (like `http` or `https`), a host (like `www.bristol.ac`), a path (like `.uk/home/maths`). Paths can have queries attached, preceded by `?` as parameters.

2 HTML - HyperText Markup Language

2.1 Tags, Attributes and, Values

Tags form the structure of HTML, with `html`, `head` and, `body` usually forming the top levels:

```
<html>
    <head>
        <title>Title<\title>
    <\head>
    <body>
        <p>Paragraph<\p>
    <\body>
<\html>
```

Attributes form parts of tags and, as expected, assign attributes to tags. This can describe the width of elements (`width`), the hyperlink attached to text (`href`) and, more:

```
<a href="www.bristol.ac.uk">Bristol<\a>
```

2.1.1 Common Tags

Below is a table containing common HTML tags:

Tag	Description
`h1`, ... `h6`	Headings
`p`	Paragraph
`br`	New line
`ul`	Unordered list
`ol`	Ordered list
`li`	List item
`em`	Emphasis
`strong`	Importance
`q`	Quote
`cite`	Citation
`var`	Variable
`code`	Source code

2.2 Block and Inline Elements

Block level elements take up the full width of the container and start on new lines, so stack vertically.
Inline elements don't start on new lines and only take up as much width as is necessary, so stack horizontally.

2.2.1 Block Tags

Below is a table containing some of the block HTML tags:

Tag	Description
`header`	This is the very top of the page
`main`	This fills the space inbetween the header and footer
`section`	This forms subsections of blocks
`div`	No meaning, for layout purposes
`p`	This forms paragraphs of text
`figure`	This forms images
`nav`	This fills the space left of the main block
`aside`	This fills the space right of the main block
`footer`	This is the very bottom of the page

2.3 Common Attributes

Below is a table containing some of the common HTML attributes:

Attribute	Description
`id`	Uniquely identifies the tag with the value
`class`	Marks tags you want to operate as a group

2.4 Forms

The form tag, shown in the example:

```html
<form method="post" action="/comment/comment.php">
    <p>
        <label for="name">Name:</label>
        <input type="text" id="name" />
    </p>
    <p>
        <button type="submit">OK</button>
    </p>
</form>
```

The method attribute takes two values GET and POST. The former places the information in the URL parameters and the latter utilises a HTTP request.
The action attribute defines an action to be performed when the form is submitted. In this case, it's sent to a PHP script.
The label for attribute should link to a input id. Additionally, the input name attribute is the key which accompanies the input value in the request.
The button tag has three types, a submit button that makes the request, a reset button that resets all fields and, a button type that does nothing by default but can be configured using Javascript.

Types can be used to make field use a specific format or be required. Additionally, they can be given placeholder text and autocompletion properties.

2.5 Escape Characters

We list these below, note that they also work in XML:

Character	HTML Representation
$<$	<
$>$	>
&	&
”	"
Non-breaking Space	

3 CSS - Cascading Stylesheets

CSS describes how HTML elements should be drawn to the screen. It can be used:

- Inline with the `style` attribute,
- Internally with the `style` tag in the `head` section,
- Externally via linking to a `.css` file.

3.1 Stylesheet Linking

We can link to external stylesheets as follows:

```
<link rel="stylesheet" href="styles.css">
```

3.2 CSS File Structure

The parts of CSS files are formed as follows:

```
selector {
    key: value;
}
```

3.2.1 Selectors

Selectors can be a:

- tag, written simply as `div`,
- class, written as `.class`,
- id, written as `#id`.

4 Encoding

Encoding is about mapping symbols to bytes. There are many standards, of which we will see a few.

4.1 ASCII - American Standard Code for Information Interchange

ASCII contains the digits 0 to 9, the lowercase and uppercase English alphabet, some punctuation and, special characters. Each of these is represented by a seven bits.

4.2 UTF - Unicode Transformation Format

The first 128 characters of UTF-8 correspond to the characters of ASCII making UTF-8 backwards compatible with ASCII. The unicode character set contains around 136,000 characters. The individual formats (UTF-8, UTF-16, etc.) encode these differently and have different memory requirements. We can choose to use UTF-8 in HTML as follows:

```
<meta charset="UTF8" />
```

4.3 CSV - Comma Separated Values

CSV use commas to separate field and `CR LF` to separate records. The record at the top is reserved (usually) for the titles of the columns.

4.3.1 Streams

We can read CSV files in as streams. Thinking about stream operations is important when considering web programming as data is usually a stream. We cannot perform operations on streams that require more than one pass (like standard deviation).

A few operations we can do are:

- filter - omitting as we go,
- map - mapping as we go,
- sum - summing as we go.

5 Representing Data

5.1 Trees

Representing data as trees requires three separators for the start and end of an item, and for fields. Additional separators are needed for quoting and escaping.

5.2 XML - Extensible Markup Language

The goal of XML is to create a straight-forward way of representing data that is machine and human readable that also can give context to data.

It allows portable, non-proprietary, hierarchical data storage. Common parsers are XPath and XQuery.

5.2.1 The Structure

XML documents are formed of five components:

- the XML declaration,
- the root element,
- attributes,
- child elements,
- text data,

illustrated below:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<labReport patientId="1234567890" specimenID="750853">
    <testResult date="2005-01-25-T12:15:37-09:00">
        <test>
            <testCode scheme="myCodes">42Hxx</testCode>
            <testName>Potassium</testName>
        </test>
    </testResult>
</labReport>
```

5.2.2 Validation

There are two validation methods, DTD (Document Type Definition) and schema. We consider the example:

```
<candidate>
    <name>Catherine Slade</name>
    <party>
        <name>Green</name>
    </party>
    <ward>
        <name>Bedminster</name>
        <electorate>9951</electorate>
    </ward>
</candidate>
```

we have the DTD validation format:

```
<?xml version="1.0"?>
<!DOCTYPE candidate [
    <!ELEMENT candidate (name, party, ward)>
    <!ELEMENT name (#PCDATA)>
    <!ELEMENT party (name)>
    <!ELEMENT ward (name, electorate)>
    <!ELEMENT electorate (#PCDATA)>
]>
<candidate> ... </candidate>
```

where PCDATA is parsed character data. Also, we have the XML Schema Definition:

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="candidate">
        <xs:complexType><xs:sequence>
    <xs:element name="name" type="xs:string" />
    <xs:element name="party"><xs:complexType><xs:sequence>
        <xs:element name="name" type="xs:string" />
    </xs:sequence></xs:complexType></xs:element>
    <xs:element name="ward"><xs:complexType><xs:sequence>
        <xs:element name="name" type="xs:string" />
        <xs:element name="electorate"
                    type="xs:nonNegativeInteger" />
        </xs:sequence></xs:complexType></xs:element>
    </xs:sequence></xs:complexType></xs:element>
</xs:schema>
```

5.3 JSON - Javascript Object Notation

JSON is a machine and human friendly data format. As it is formed by text, it can be parsed and generated by most programming languages and can be transmitted easily.

5.3.1 Comparisions to XML

Here are some key differences:

- JSON allows arrays,
- JSON tends to be shorter,
- JSON is quicker to read and write,
- JSON can be parsed by standard functions.

6 Databases

6.1 Web Architecture

A multitier architecture or n-tier architecture is a client-server architecture which physically separates presentation, application processing and data management functions.

A common example is the 3-tier architecture which is formed by presentation, application and, database layers.

6.2 SQL

SQL tables are formed by these main components:

- Columns / Fields / Attributes,
- Rows / Records / Tuples.

6.2.1 Super Keys

A super key is a combination of the fields of a table such that using just those columns, we can uniquely identify each record.

6.2.2 Candidate Keys

A candidate key is a minimal super key.

6.2.3 The Primary Key

The primary key is a chosen, 'most important', candidate key.

6.2.4 Useful Commands

There are many useful (MariaDB) SQL commands:

Command	Description	More information
`CREATE`	Creates a table	
`DROP`	Deletes a table	
`TRUNCATE`	Deletes all records in a table	
`SELECT`	Picks values from a table	Use * to select all
`INSERT`	Inserts a record into a table	
`DELETE`	Deletes all records in a table	Usually used with a `WHERE` clause
`UPDATE`	Updates values in a table	
`AUTO_INCREMENT`	Automatically increments and assigns a field	
`--`	Initiates a comment	
’’	Used for strings	
‘‘	Used for database values	

6.2.5 Exporting and Importing

Using mysqldump we can export a database using the following command in the MySQL client command line (with a following example):

```
mysqldump -u student [options]
    dbname > filename.sql

mysqldump --skip-lock-tables
    --add-drop-table dbname > filename.sql
```

We can similarly use the client to import a database:

```
mysql -u student dbname < filename.sql
```

6.3 Relational Databases

In relational databases we have tables (relations) formed by rows (tuples) and columns (attributes) containing data.

6.3.1 Entities

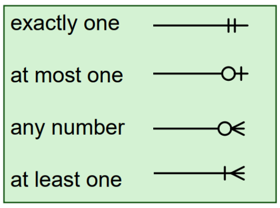
When forming a database, we have to consider the entities that are of interest. Entities have attributes (generally, if an attribute is referred to by multiple entities, it should be its own entity).

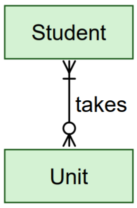
6.3.2 Keys

Keys identify instances of our entities. Considering candidate keys (a minimal key uniquely identifying entities), if it spans multiple attributes we say it is composite. Additionally, an artifical attribute generated for the sole purpose of being a primary key is called a surrogate key.

6.3.3 Relationships

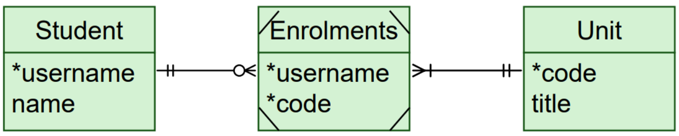
Relationships associate types of entities. We use Crow's Foot notation:





The example on the right describing the fact that a unit is taken by at least one student and each student takes any number of units.

We can introduce intermediary entities that describe relationships between entities.



Note how the new entity is unique to the pair of entities it describes the relationship between.

6.3.4 Forming Tables from Relationships

Entities become tables, attributes become columns, entity instances become rows.

For unique relationships between entities, we can use a `UNIQUE` foreign key relating them. If two entities require each other, we merge their tables. Otherwise, wherever the relationship is mandatory, we place the key (with the property `NOT NULL`) or if it isn't mandatory at all, we can use the property `NULL`.

For one-to-many relationships, we place the foreign key on the 'many' side of the relationship.

For many-to-many relationships, we create a new table (a 'join' table) containing two foreign keys to the original tables (with its primary key being the composite of these two foreign keys). This table contains pairs of IDs from the original tables to describe relationships.

6.4 Projection and Selection

Projection is about selecting columns from tables whereas Selection is about selecting rows from tables. While projecting, we can perform operations on our columns where it makes sense (concatenating first and last names, adding one, etc.). While selecting, we can provide constraints on what records are selected.

6.5 Products and Joining

6.5.1 Inner Joining

To get the cartesian product of two tables we can use the following command:

```
SELECT * FROM TABLE1, TABLE2 WHERE TABLE1.id = TABLE2.id;
```

This pairs each record in `TABLE1` with a record in `TABLE2`. We can write this using the `INNER JOIN`:

```
SELECT * FROM TABLE1 INNER JOIN TABLE2 ON TABLE1.id = TABLE2.id;
```

and can stack this calls too:

```
SELECT * FROM TABLE1
    INNER JOIN TABLE2 ON TABLE1.id = TABLE2.id
    INNER JOIN TABLE3 ON TABLE2.id = TABLE3.id
    ...
WHERE ...;
```

If the two tables have a field with the same name, we can use `JOIN USING`:

```
SELECT * FROM TABLE1
    INNER JOIN TABLE2
USING (id);
```

In fact, using `NATURAL JOIN` SQL will search for an identical column automatically:

```
SELECT * FROM TABLE1
    NATURAL JOIN TABLE2;
```

6.5.2 Outer Joining

We may potentially join while referencing rows with `NULL` content. This is where we use the `LEFT`, `RIGHT` and, `FULL OUTER JOIN`s. The `LEFT` outer join ensures that each record from the left table appears in the result and similarly for the `RIGHT` outer join. The `FULL` outer join ensure each record from both tables appears at least once.

6.5.3 Self Joining

We can join a table to itself, for example - we may want to find all pairs of field subject to a constraint on one of their attributes. This can be done using the `INNER JOIN`:

```
SELECT * FROM TABLE1 T1
    INNER JOIN TABLE2 T2
    ON ...
    WHERE ...;
```

6.6 Set Operations

The set operation commands have the form:

```
SELECT ...
    [UNION [ALL], INTERSECT, EXCEPT]
SELECT ...;
```

They all behave as expected, the extra `ALL` condition on the `UNION` command ensures that duplicates are not removed.

6.7 Summarising

6.7.1 Unique Records

When selecting data, we can specify that we want the result to be `DISTINCT`:

```
SELECT DISTINCT ... FROM ...;
```

6.7.2 Summing Records

When selecting data, we can return the number of rows in the resulting table with `COUNT`:

```
SELECT COUNT(...) FROM ...;
```

By using `COUNT` in conjunction with `DISTINCT` can be more informative. Additionally, we can use * to count all the rows in a table.

`COUNT` always returns a single value (in a row) and ignores `NULL` values.

6.7.3 Grouping

We can use `GROUP BY` to associate `COUNT`s with other values in our table:

```
SELECT key, COUNT(*) FROM ... GROUP BY key;
```

In fact, `COUNT` is what we would call a 'aggregate' function - taking in a list of values and returning a single value - and we can replace `COUNT` with other aggregate functions like the average, maximum, minimum and, median.

We have that `WHERE` clauses must appear before aggregation, so cannot refer to aggregates. However, we can use `HAVING` which is capable of refering to aggregates and column aliases exclusively.

6.8 Subqueries

We can utilise subqueries by putting queries in parentheses in place of values in queries. We can use:

```
WHERE ...
    [
        [NOT] IN,
        [NOT] EXISTS,
        ANY,
        ALL,
    ];
```

with subqueries to filter quickly.

6.8.1 Views

We can use `CREATE VIEW` to create a reference to a select statement which will act like a table:

```
CREATE VIEW name AS
    SELECT ... FROM ...;
```

6.9 Execution Order

The execution order of SQL queries is detailed below:

```
SELECT ...
FROM ...
... JOIN ...
WHERE ...
GROUP BY ...
HAVING ...
ORDER BY ...;
```

6.10 Normalisation

This eliminates redundancy and dependency which causes anomalies when inserting and deleting data. This causes the database to become more complicated but allows operations on it to be performed more easily.

The normal forms are numbered and cumulative.

6.10.1 1NF - The First Normal Form

A schema is in 1NF if there are no:

- Collection-valued attributes,
- Repeated attribute groups, as in duplicate information in different tables,
- Duplicate rows.

6.10.2 2NF - The Second Normal Form

A scheme is in 2NF if it is in 1NF and there are no partial functional dependencies from a candidate key to a non-key attribute. This means an attribute should not be uniquely defined by a subset of the candidate key.

6.10.3 3NF - The Third Normal Form

A schema is in 3NF if it is in 2NF and has no transitive functional dependencies. This means there are no set of three attributes $\{A, B, C\}$ where B and C are non-key and:

$$A \to B \to C,$$

meaning A is functionally dependent on B which is functionally dependent on C. As it doesn't matter if A is key, we can just consider functional dependencies between two non-key attributes and just use our candidate key as A. We can see that the consequence of this is that all non-key attributes depend only on the candidate keys.

When there is such a transitive functional dependency, we can decompose the table into two tables without losing any information (lossless decomposition).

6.10.4 BCNF - Boyce-Codd Normal Form

A scheme is in BCNF if it is in 3NF and the left side of each non-trivial functional dependency is a superkey.

6.10.5 4NF - The Fourth Normal Form

A schema is in 4NF if it is in BCNF and for every multi-valued dependency, the dependency is trivial or the left side is a superkey. A multi-valued dependency is where when we insert a new record, insert multiple to satisfy the structure of the table.

6.10.6 5NF - The Fifth Normal Form

A scheme is in 5NF if it is in 4NF and all decompositions of it are not lossless.