Week Two: Databases & Design

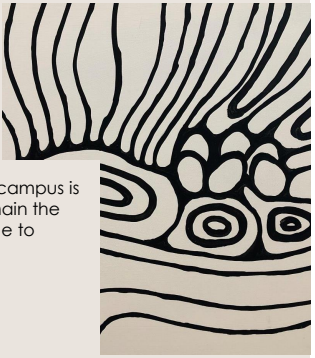
INMT5526: Business Intelligence

Semester One, 2024

Tristan W. Reed
UWA Business School

Acknowledgement of country

The University of Western Australia acknowledges that its campus is situated on Noongar land, and that Noongar people remain the spiritual and cultural custodians of their land, and continue to practise their values, languages, beliefs and knowledge.



Artist: Dr Richard Barry Walley OAM

What is a database?

- Databases allow us to store, retrieve, change and analyse data in a structured format, enabling us to answer questions to make business decisions.
- Before we go further, we should consider *what a database is*. Things that we may want to consider are:
  - What makes a database a database? What are the vital components?
  - Conversely, how do we know something *isn't* a database?
  - Can databases only take one format, or can they take many?

(What are) databases?

- In simple terms, a database is just a group of *organised data*.
- This simply means that there is structure to the data.
  - It is stored in a regular and (somewhat) consistent way.
  - Think tables, lists and the like – rather than a Word document.
  - Tabular data is but the basis of one form of database – albeit the most common.
  - But doesn't that just mean it is a spreadsheet? Not exactly…

Evolution of databases

- Much like *Business Intelligence*, databases themselves had their genesis in the 1960's – a time where computing was much simpler than today.
  - This is primarily due to limited resources, driven by a lack of technological sophistication.
- The first databases followed the *navigational* architecture.
  - Where a tree-like hierarchy existed of parent and child relationships and relationships could only be one-to-many in nature (a parent has multiple children).
  - Although these systems organised data, their simplicity led them being too inflexible to solve every data storage and modelling problem.

Database entities

- An *entity* is *something* that we store data about, in terms of its *properties*.
  - This could be a real world thing (a person, a location, a machine, a physical product) or something (at least some)what intangible (a system, a conceptual product).
  - We consider the entity in terms of its *properties* – the things that make the *entity* uniquely what it is and describe it, which are present in each of the entities.
  - The values of the property differ between entities, but not the presence of the property.

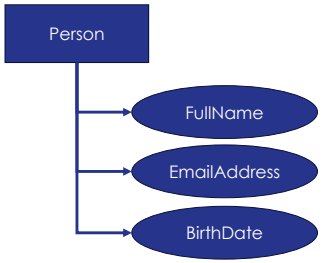
Database Design

- While we are thinking of *entities* rather than data elements themselves, consider a real-world relationship between *things*.
  - Generally, these are in the form of parent-child – but not always and there are others.
  - However, consider those that can't be modelled as a parent-child relationship.
  - An example of a *parent-child* relationship would be an Employee is a Person.

Entity relationship diagrams

- We can represent this relationship in terms of a *entity-relationship diagram*.
  - These diagrams visually show the elements (properties and entities) and relationships between them.
- There are many different ways to create entity relationship diagrams, we will show just but one – which isn't sayings its 'the best'.
  - Different ways to show different information - the elements and relationships.

An example of the diagram



How do we decide what goes in here?

Structured Query Language (SQL)

- SQL is a language used with *relational* databases to manipulate data.
  - Some consider it a programming language – but is it? Consider PL/SQL & extensions.
  - Common tasks include creating, viewing, changing and removing data.
  - Can also manage users, define models and other administrative tasks.
- In some contexts, SQL has been replaced by *object-relational models*.
  - These allow the use of databases in the same way as objects in programming languages, thereby making the development of user-facing systems easier.

Databases vs spreadsheets

- The main difference between databases and spreadsheets are that spreadsheets are designed for simple data manipulation tasks undertaken by a single user (at least a single user at a time).
  - Databases can handle multiple people interrogating the data at the same time;
  - Databases provide greater performance for larger data sets;
  - Databases also provide finer-grained access controls.
- Can you think of a benefit a spreadsheet has over a database?

Data Model Formats

- Traditional (and most common) *relational* databases model data in terms of *tables* of *rows* and *columns*.
  - Other data models exists that don't do this – e.g. NoSQL document databases;
  - A database can contain many tables of 'infinite' rows and columns, but there are practical and logical considerations to this.
  - This format is generally used due to the efficiency and regularity of the format – it is known each row has the same columns and vice versa.
  - We can manipulate these with SQL and a database management system (DBMS).

Rows and columns

- In a relational database, one or many *tables* exist with *rows* and *columns*.
  - Generally, the "rows" refer to observations or records – each one identifying an individual or unique occurrence of an activity, phenomena or 'thing'.
  - The "columns" refer to the attributes of the table – or what is being measured.
- Tables can contain many attributes that are related or sometimes seemingly unrelated attributes (sometimes in groups of attributes).
  - The process of *normalisation* is undertaken to ensure the attributes in each table are all related to minimise the duplication of data.

Other database structures

- A *relational* database format is not the only one that exists. There are many others that depending on the situation, may be more suitable:
  - *Object-oriented databases:* models objects (instances i.e. 'rows') of entities or 'things' (e.g. products, employees, stocks) that have (unique values for) attributes or properties (analogous to the 'columns').
  - *Distributed databases:* these are databases that are represented as data physically over more than a single place (machine) or location. What is a benefit of this? Is it better or worse than having data in a single location? Are any problems introduced?

Other database structures

- Continuing on from the last slide...
  - *Data warehouse:* a term that refers to a central data repository that is designed for quick retrieval of data for analysis. This is really the opposite of a *distributed* database.
  - *NoSQL:* referring to "not only SQL" rather than "not SQL" (although termed by some as 'non-relational'), these databases allow less structured (unstructured or semi-structured) data to be stored and queried. A well known example is MongoDB, which is used with many (JavaScript-powered) web applications. Can you think of an issue with this style of database when compared to a traditional relational-style one? How about a benefit?

Other database structures

- Still continuing one for a little while…
  - *Graph databases:* these store data in terms of entities (individual 'things') and the relationships between these (both the entities related and how they are related).
  - *OLTP databases:* an 'old-fashioned' term for "online transactional processing" database, in simple terms, this is a database which is designed for many users and operations being undertaken simultaneous – think of it being designed for 'performance'.
  - *Open-source databases:* such as MySQL which we are using, where the underlying source code powering the DBMS is available for reuse and modification – unlike Oracle/MS.

Other database structures

- Last one of these (let's hope!)…
  - *Cloud databases:* simply a database stored in the cloud. This primarily deals with the fact the data can be easily distributed and management and maintenance of the underlying system abstracted away from the database administrator.
  - *Multimodel database*: a fusion of (some of) the above data formats, ideally yielding the best benefits of each.
  - *Document (JSON) database:* collections (groups) of JSON documents (key-value) are stored and queried, rather than relational tables.

Database software

- Many pieces of software exist to manipulate databases – some of which use graphical interfaces and others use text-based interfaces.
  - We have looked at "MySQL Shell" ourselves so far – with a text-based interface.
  - Common tasks that this software can undertake include data retrieval and manipulation, model definition and modification, backup, reporting, user access and security.
  - This software allows us to complete simple tasks and helps users ensure that the tasks are completed correctly – especially important in terms of data integrity and security.
  - Benefits exist between graphical and text-based interfaces – not one is better!

Database management systems

- A database management system (DBMS) is the interface between the database itself and the end-user, allowing them to interact with it.
  - This allows them to run the commands and tasks needed to manipulate data or perform administrative tasks.
  - Sometimes the "end user" is not a physical user, but another computer or machine running software to integrate with the system.
  - Often, the database and its management system are considered "one and the same" as the database can't be used without it.

MySQL databases

- Many relational DBMS exist that work similarly and achieve the same aim.
  - "Oracle", PostgreSQL, MSSQL (Microsoft SQL), MySQL, MariaDB, Amazon…
- We are using the very popular MySQL, for a few reasons…
  - It works (both client and server) on many different platforms.
  - Free to download and use, flexible, powerful and efficient.
  - Used by many popular web applications such as Airbnb, LinkedIn, YouTube and Twitter (not sure how true this is) so it is somewhat ubiquitous and likely very reliable.

Business decision making & DB's

- Recall why we are here in *Business Intelligence*?
  - Our aim is to learn skills, tools and techniques to help us make business decisions.
- These days, businesses are connected to (and generating) more and more data as the days go by.
  - Sensor systems (such as Internet of Things) generate mass amounts of data that needs analysis to be able to pick up patterns and insights;
  - Rather than just storing data, databases can be used to help analyse data at a large scale and from multiple systems.

Business decision making & DB's

- Together with BI tools (which can be linked to databases), the power of today's computer systems can allow organisations to analyse this data.
  - This can be used to help businesses run more efficiently, make better decisions and react to changes quicker (agility) and grow (scale) quicker.
  - Decisions can only be make quickly "in time" when the storage, access and analysis of the data is easy, consistent and fast - otherwise by the time the analysis is completed, the insights or decisions can be out of date. Hence, the DBMS used must be performant.
  - Volumes of data just continue to grow and grow – where do we go next? Web 3.0?

Self-driving databases

- "Self driving" databases use AI/ML to automate many database management tasks, making databases somewhat "autonomous".
  - This makes it easy for businesses to run and manage a database, as it is maintained automatically as the system runs tasks when it deems they are needed.
  - Arguably, the cloud provides this to a limited extent – however "true" self driving databases provide the ability to secure and repair themselves.
  - This should lead to lower costs as less database administrators needed, improved security due to the automated application of "best practice" and finely tuned performance.

Database challenges

- Many common issues exist within databases, which are important to managers and users of these databases and affect businesses intimately:
  - Large increases in data volume from new connected machines, sensors and various other sources. Not only is this a performance issue, but the data must also be organised, integrity must be assured and managed.
  - Security of private and personal data to ensure legal requirements are met and to prevent (costly and reputation-damaging) data breaches, while also balancing the ease-of-access to authorised users (consider users' right to their own data).

Database challenges

- Continuing on from before…
  - Ensuring the demand of end-users to access and manipulate information in a timely manner can be met to ensure business decisions can be made 'in time';
  - Managing database systems with an increase in threats requiring timely updates to patch these alongside 'tuning' performance to ensure the system is running at its best;
  - Running into limits of scalability and planning scalability in advance when it is difficult to predict utilisation of database systems;
  - Various data sovereignty, performance or other contractual/legal requirements.

Database keys

- "Keys" are used in relational databases as an attribute to uniquely identify individual observations (rows) from each other.
  - As such, the values must be unique – otherwise more than one row can be identified!
  - Primary keys (PK)'s are stored within a table, whereas a Foreign key (FK) is a reference to a PK in another table other than the current table.
  - This therefore creates a relationship between the two tables such that the second table can be looked up from the first and considered to 'extend' the values of (and the) attributes defined within it.

Database keys

- Generally, these keys are automatically increasing numeric values, however they can theoretically be of any format.
  - Within reason: generally these are only text-based or integer-numeric formats.
- However, as these keys can be used to look up particular rows, performance must be considered when using esoteric formats.
  - Imagine using an image as a primary key! It would be hard in some cases to tell apart.
  - Indexes are built for searching from keys, which is difficult in other formats to do.

A relationship is born

- If the value of a foreign key in a row of one table defines a row in another table, a relationship between two tables is created.
  - How we term this relationship depends on how it is formed.
- The various *cardinalities* of relationships (and their abbreviations) are below:
  - *One to Many (1:M)*: One record in a table is linked to many records in another table.
  - *Many to One (M:1)*: Many records in other tables are linked to one record in a table.
  - *One to One (1:1)*: One record in one table is linked to exactly one record in another.

More relationship types

- Many to Many (M:M) relationships are common, but the least structured and most complex to deal with.
  - Many rows in one table link to many rows in other (or the same) tables (and vice-versa).
  - In practical terms, generally a "linking table" (or multiple) is needed that states the PK's of the two tables that are linked together.
  - A graph database may be more suitable for M:M use cases.
- Some also define relationships in terms of '0' as well - not too helpful!
  - If anything, this tells us NOT to create a relationship!

Relationships in ER diagrams

This part is shown in many, many different ways – diagrams, lines, labels.

Designing a database

- To design the database, we must complete the following tasks:
  - Define the sets of entities for the system;
  - From this, determine the Tables that are required;
  - (Define and) identify the Primary and Foreign keys;
  - Describe the cardinality of each relationship.
  - (Define and) identify the attributes.
- Consider *what* and *why* we are doing this – what can we derive?

Design of databases

- Design is important – if we get this wrong, our database may not meet our needs and the exercise of creating a database is futile.
  - Think back to last week and the issues that we had discussed then.
- We can now design databases without using a database management system.
  - However, this is not the only (or best) way to define what databases should look like.
  - Practice helps us get better at this task and helps us think about the problems.

THE UNIVERSITY OF WESTERN AUSTRALIA

Questions and Answers

tristan.reed@uwa.edu.au