SQL and Databases Inlämningsuppgift – Mark Peters

1 A) **DBMS, 'Database Management System'**, is a software system or collection of systems of various complexity and composition that function as an interface (or intermediary) between a database and an end-user. It allows users to create, read, update and delete data (CRUD) in a database.

B) **Redundancy** occurs when data is duplicated unnecessarily in a database, if this data is not connected it may lead to inconsistencies and severely undermine the reliability of the stored information. These difficulties can be resolved through reconciliation, integration, and normalization operations, but these are costly and can be avoided through careful database planning and execution.

Redundancy can be positive if the goal is to duplicate the information for safety and security reasons.

C) **Normalisering tom 3NF – (Third Normal Form)**

A database relation is said to meet third normal form standards if all the attributes (e.g. database columns) are functionally dependent solely on the primary key. To achieve third level form, one must have the second level in place. There must, also, not be any transitive relationships, such as where instead of A determining C directly, it goes through an intermediary of B (A --- B, B --- C). In these cases, one can encounter update anomalies where updates may not apply across the board or delete anomalies where the deletion of a single value would lead to the deletion of a parent value.

The main point is that each value must have a relation to a key, and these keys must work together to keep uniformity across the whole database.

D) AVG. This is an inbuilt SQL function that returns the average number from a list called by a 'select' command.

Question 2:

SQL is a relational model that uses tables in a database that are connected through keys and commands, No SQL is a document or graph-based model which was created due to the decreased costs of data storage.

SQL is easy to set up and is compatible with many different programs and systems. It is high-performance and is extremely effective at structuring data. On the negative side, it is difficult and time-consuming to design and setup an effective database, and so can be inflexible when attempting to scale-up.

No SQL is cheap to develop and implement and can be developed quickly and with precision. In general, it is a faster option than SQL and it works well with cloud-based applications. Its drawbacks are that it is unsuited to interconnected systems and its technology is still developing and expanding, so it will cycle through many iterations before it reaches its optimal stage.

Question 3:

JSON is a simple, pared down text formatted script which is derived from Javascript object notation. It stands for JavaScript Object Notation and is used primarily as a vehicle for sending data between computers.

XML might be "old" and complex, but its complexity is what enables it to not only transfer data but also to process and format objects and documents. Another great advantage of XML is that it handles comments, metadata, and namespaces, none of which JSON can do.

Question 4:

Data Integrity is a counterpoint to ‘Redundancy’. It is achieved when the accuracy and consistency of a database can be maintained over its lifetime. The keys to preserving data integrity are the validation of data and of inputs, the removal of duplicate data, the control of those who have access to data (re: SQL injections) and being able to track users and abusers through ‘audit trailing’.

To maintain integrity means securing the data from corruption and protecting it from being destroyed. There are different storage systems, such as RAID, which use mirroring or striping techniques to duplicate or network data. This, along with logical database systems, contribute to safe and dependable systems.

Question 5:

Simply put we can build **SQL servers** either vertically or horizontally. Vertical servers are built by adding raw power to a system, by adding a larger RAM or CPU to a server to manage an increased data load. Horizontal scaling means adding more instances or nodes together so that their collective power is greater than their individual.

A CDN (content delivery network) would be used for the quick dissemination of large media files such as are common currency with sites like YouTube and Instagram.

Question 6:

A **table** is a collection of logically organised data where each row represents a unique record, and each column represents a field in the record. A **view** is a virtual table formed from fields from one or more tables in a ‘real’ database. A **view** can give the user real and up to date information as it is reformed every time the query is run.

Question 7:

The main difference between an **application** and a **database** is that a database will be relevant so long as you have a computer to read it, while applications come and go with almost unceasing rapidity. Access control, data security, and data integrity rules should reside where they will have the longest lasting effect – right next to the data. In that way, applications can change their shape, role and function but the data they communicate with will still have the same consistency and reliability that it always had.

Laser Videos Database - Motivation

Many years ago, when I was still a young man and eager to get into the movie industry in any way shape of form, I took a job working in a specialist video store in Dublin. Everyone who worked there was a film expert and snob – if you've ever seen 'Clerks' you'll recognise the type – and I was no different. Even though we only worked in a video store we acted like we were the arbiters of taste in all things film.

When thinking about what kind of database I wanted to create I could think of nothing better than recreating the kind of technology that we used way back in the late 90's, early 00's.

Database – Design

1. Films
film_id
film_name
rental_rate
genre
Rating
language
description

8. Film/Actor
film_id
actor_id

7. Actor
actor_id
first_name
last_name

2. Customers
customer_id
first_name
last_name
email_address
phone_number
date_created

4. Payment
payment_id
customer_id
staff_id
rental_id
amount
payment_date

6. Inventory
inventory_id
film_id
date_entered

9. Staff ID card
staff_id
full_name
work_login
work_logout

3. Rentals
rental_id
inventory_id
date_rented
date_returned
customer_id
staff_id

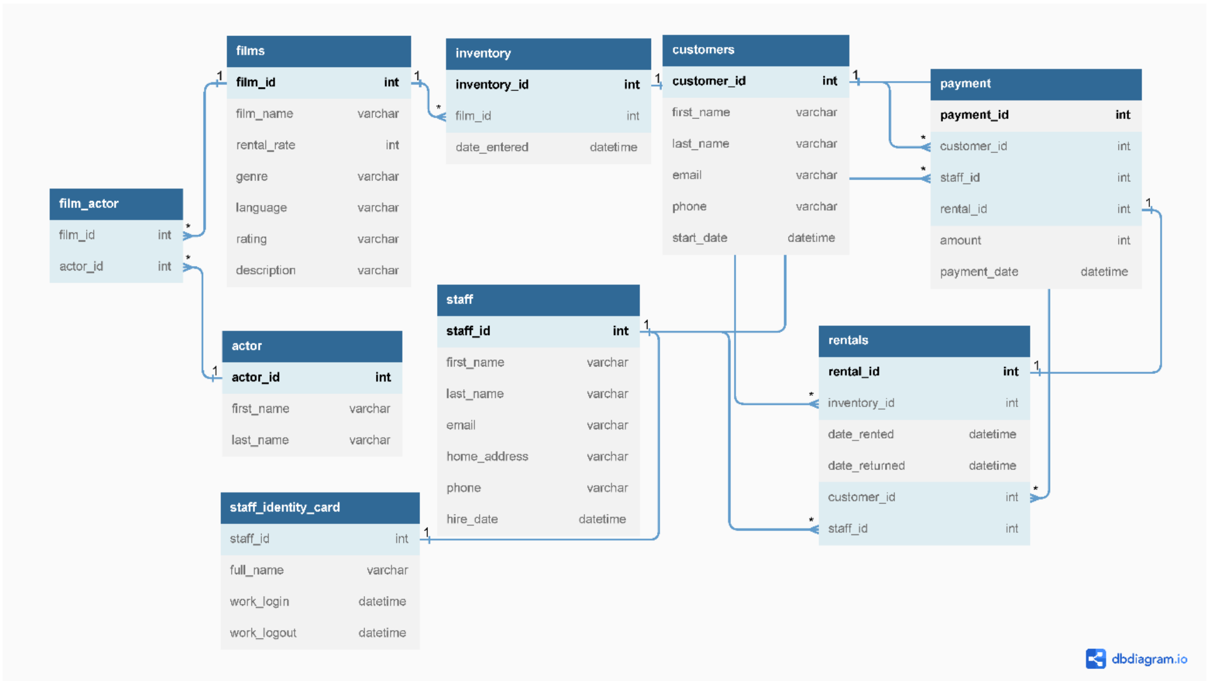
5. Staff
staff_id
first_name
last_name
email_address
home_address
phone_number
date_hired

The database is divided into nine different tables in total: Films, Customers, Rentals, Payment, Staff, Inventory, Film/Actor, and Staff ID Card. '**Films**' is a table of all the films the store can buy copies of, there are a total of 23 films. The store bought 2 – 3 copies of only the first 16 titles on the list (before they ran out of money 😊). These copies were entered into the '**inventory**' table. The '**customer**' table has a collection of 20 members containing personal information as well as the date that they became members. Separate '**actor**' and '**actor/film**' tables were also created, the reason for this will be explained later.

To manage the daily running of the store a **staff** of five have been hired, each staff member gets a '**staff ID card**' which logs both when they start and when they finish work. Staff members register each film rental into the '**rentals**' table, the date when rentals occur and when films are dropped back, are logged into this table. A separate '**payments**' table records what each customer spent on each transaction (more items can be paid for than just the rental of videos).

Each table has single or multiple relationships to the other tables in the database – either 'one to one', 'many to one' or 'many to many'. They all have a primary key except '**Film/Actor**' whose purpose is to allow more than on actor to be associated with the same film, or for one actor to be connected to many films. If '**Actor_id'** was given a column in 'films', then only one actor could be attached to each subject film. It is the foreign keys in the tables that establish the kind of relationship they have with the rest of the database.

The **Staff ID card** table exists simply to add a 'one to one' relationship to the list, though in reality it would serve a real purpose by keeping track of when staff members log into and out of work.



There are many relationships contained within the database and they are best illustrated in the above picture, but I will give an example from each of the main groups:

- **1 -1 relation:** Staff to staff ID card, each member of staff can have only one staff card and each card belongs to only staff member.
- **1-many relation:** this is the most common type of relationship in a database and that is true for the Laser Videos database also. One example of this kind is between customers and rentals; each customer can rent many films over the course of their subscription.
- **many-many relation:** the actor/film table is an example of 'many to many' in that each actor can appear many times in the films table, and each film could conceivably appear many times in the film table.

8) Modify a SQL database:

Within the homework folder can be found a query file containing all the queries that were made to modify the database in multiple ways.

There are multiple examples for each element in the **CRUD** list.

- There are three examples of **creation** queries; there is the addition of a column to an existing table and two examples on inserting data into different tables.
- There are two **reading** of data examples, both containing a 'where' condition and one with an additional 'order by' condition.

- There are three ‘**update’** examples which show changes in values in a column. One of these updates uses an ‘IN’ function.
- There are two ‘**delete’** examples. The first example deletes a group of repeated rows from the rentals table. The second deletes a column that was introduced earlier for demonstration purposes.

I used a ‘**Left Join’** to add the inventory table to the films table, and then filtered for the columns which had a ‘null’ value for’ inventory_ID’. This produced a list of films which was **not** in the store’s film library.

I then used a double ‘**Inner Join’** to connect the ‘many to many’ tables of actor, films, and actor/film to get a comprehensive list of which actors were in which films.

I wrote a **‘nested query’** to produce the same list as the ‘left join’ query above. The second nested query searches for the customer details for those who made payments during a half-month period.

A **‘Group By’** is used to group payments made over a two-month period. I used SUM to add up the transactions and the AS function to rename the sum column.

I used **@variables** to search the customer table for a member’s information. Such variables are the most common way that user data is inputted to find information in any database.

Lastly, I adapted a program that you gave us in a lecture to create a **procedure** that could find all the movies in the database in a genre. This makes use of @variables, procedures and an **EXEC command**.