SQL - Basics

Contents

What is Data

First of all, what is data? This course covers Data Science after all, we need to know a brief definition of data

- **Data** can be thought of as recorded measurements of something in the real world. This *something* is a *unit of observation.*
    - *For example, a list of people's height is data.*
    - *A person would be the unit of observation **(sample)***
- Data can describe a vast amount of different things.
    - *For example, there is a lot of data we can use to describe a person.*
    - *Each measurement is called a variable **(feature)***
    - *Each observation (the hight of a person) is a **data point***
- Several data points together create a ***Dataset***

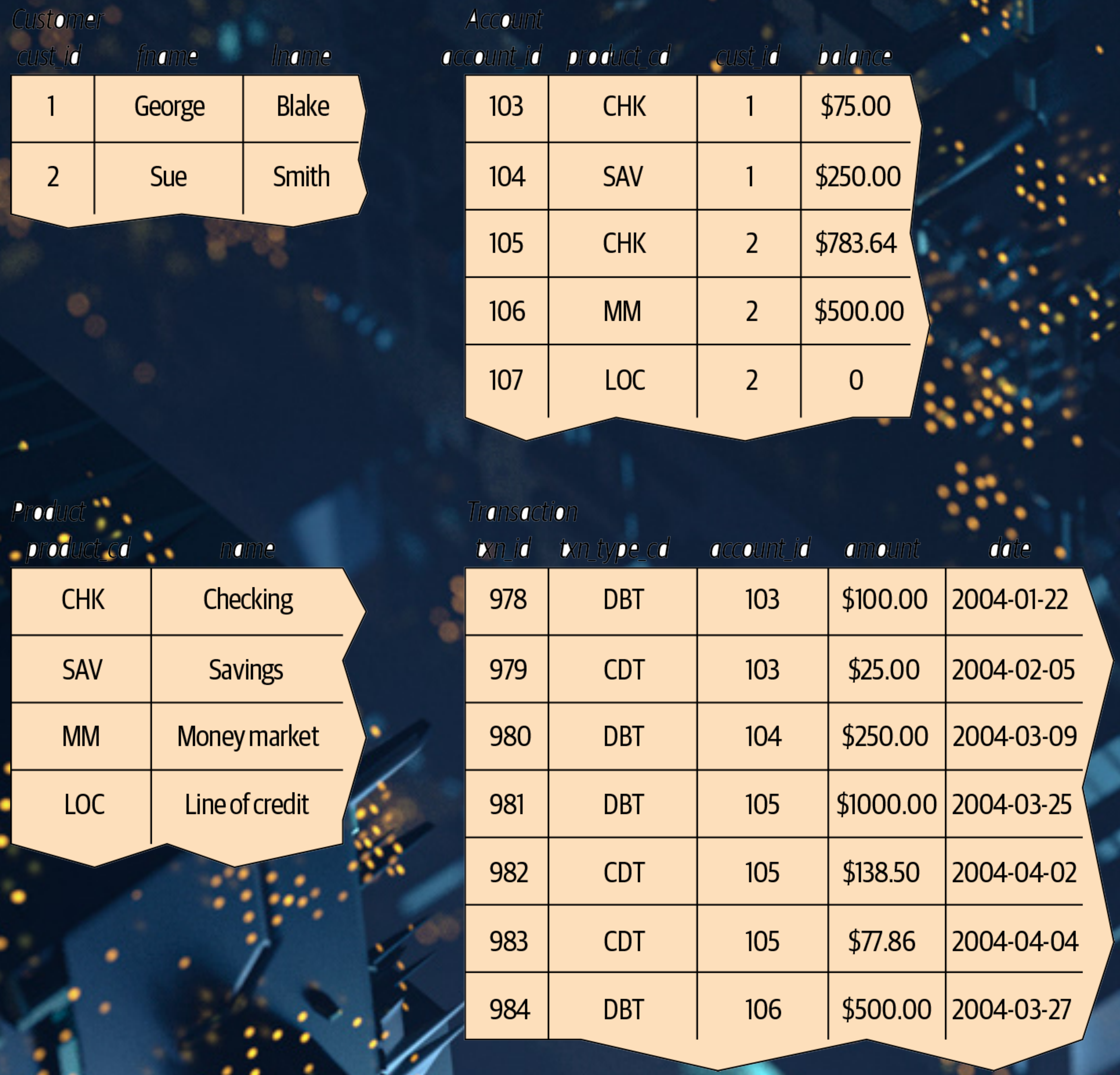
Relational Database

Data is useful to obtain valuable information. We could analyze now that data by hand, but of course, computers can analyze much more information than humans. There are two tools for storing, organizing, and processing data in a computer. The first one is:

- Relational Database:
  - Utilizes the relation model of data: ***Redundant data is used to link records in different tables***
  - Data is organized as relations, containing a ***relation key*** *(ID)*
  - *Relations are usually implemented as **Tables***
  - *Tables are assimilated in common collections in databases called **Schemas***
    - *For example: One Table contains (ID, Name, Last Name, Age)*
    - *Another Table contains (ID, Height, Smoker?)*
    - *And another Table contains (Smoker?, Cancer Development)*
- The software used to manage relational databases is referred to as a ***Relational DataBase Management System (RDBMS)***

Relational Database

- ***Redundant data is used to link records in different tables***



Customer

cust_id	fname	lname
1	George	Blake
2	Sue	Smith

Account

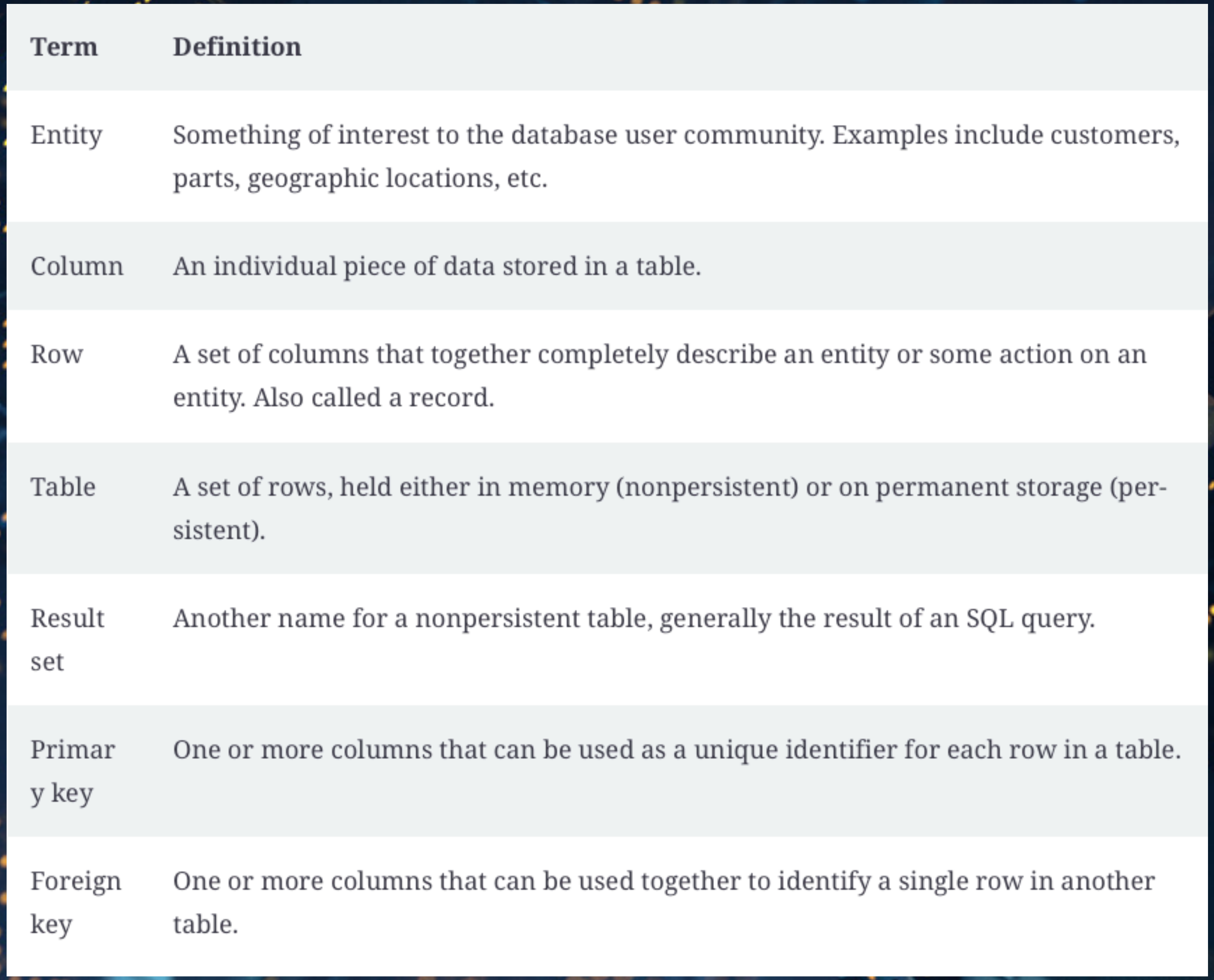
account_id	product_cd	cust_id	balance
103	CHK	1	$75.00
104	SAV	1	$250.00
105	CHK	2	$783.64
106	MM	2	$500.00
107	LOC	2	0

Product

product_cd	name
CHK	Checking
SAV	Savings
MM	Money market
LOC	Line of credit

Transaction

txn_id	txn_type_cd	account_id	amount	date
978	DBT	103	$100.00	2004-01-22
979	CDT	103	$25.00	2004-02-05
980	DBT	104	$250.00	2004-03-09
981	DBT	105	$1000.00	2004-03-25
982	CDT	105	$138.50	2004-04-02
983	CDT	105	$77.86	2004-04-04
984	DBT	106	$500.00	2004-03-27



Term	Definition
Entity	Something of interest to the database user community. Examples include customers, parts, geographic locations, etc.
Column	An individual piece of data stored in a table.
Row	A set of columns that together completely describe an entity or some action on an entity. Also called a record.
Table	A set of rows, held either in memory (nonpersistent) or on permanent storage (persistent).
Result set	Another name for a nonpersistent table, generally the result of an SQL query.
Primary key	One or more columns that can be used as a unique identifier for each row in a table.
Foreign key	One or more columns that can be used together to identify a single row in another table.

SQL - A brief intro

"Along with Codd's definition of the relational model, he proposed a language called ***DSL/Alpha for manipulating the data in relational tables****. Shortly after Codd's paper was released, IBM commissioned a group to build a prototype based on Codd's ideas. This group created a* ***simplified version of DSL/Alpha that they called SQUARE****. Refinements to SQUARE led to a language called SEQUEL, which was, finally,* ***shortened to SQL****. While SQL began as a language used to manipulate data in relational databases, it has evolved […] to be a* ***<u>language for manipulating data across various database technologies</u>*** *[…]*

One final note: SQL is not an acronym for anything (although many people will insist it stands for "Structured Query Language"). When referring to the language, it is equally acceptable to say the letters individually (i.e., S. Q. L.) or to use the word sequel."

Extracted from 'Learning SQL' by Alan Beaulieu

SQL - Pros and Cons

SQL Databases provide a ton of ***Advantages*** that may it the de facto choice for many applications

- **Intuitive:** Relations that almost anyone can understand
- **Efficient:** They use normalization so it doesn't repeat its representation (less space)
- **Declarative:** You tell the data that you want, and the system takes care of how to execute the query
- **Robust:** Most databases have the ACID compliance (Atomicity, Consistency, Isolation, and Durability), guaranteeing the validity of data even if the hardware fails

SQL - Pros and Cons

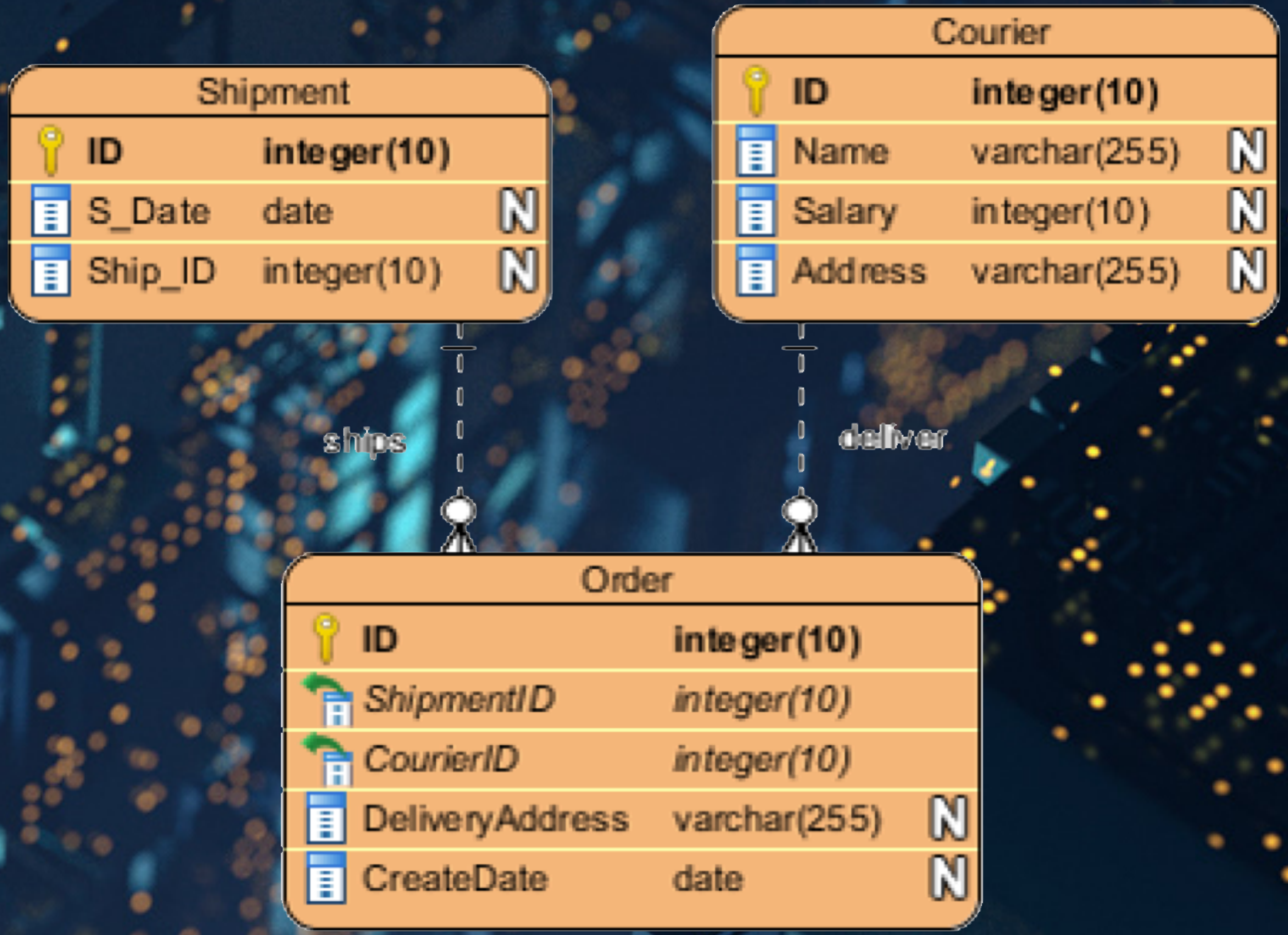
However, we might find some ***Downsides*** when working with SQL databases:

- **Lower specificity:** Sometimes, SQL's functionality is limited to what it has been programmed to do. Not common though, since it management systems get updated
- **Limited Scalability:** Due to the robustness might be an impediment for scaling data.
- **Object-relation mismatch impedance:** Sometimes objects have attributes with many-to-many relationships. For example, a costumer may own multiple products, but each product may have multiple objects

SQL - Entity Relationship Diagram

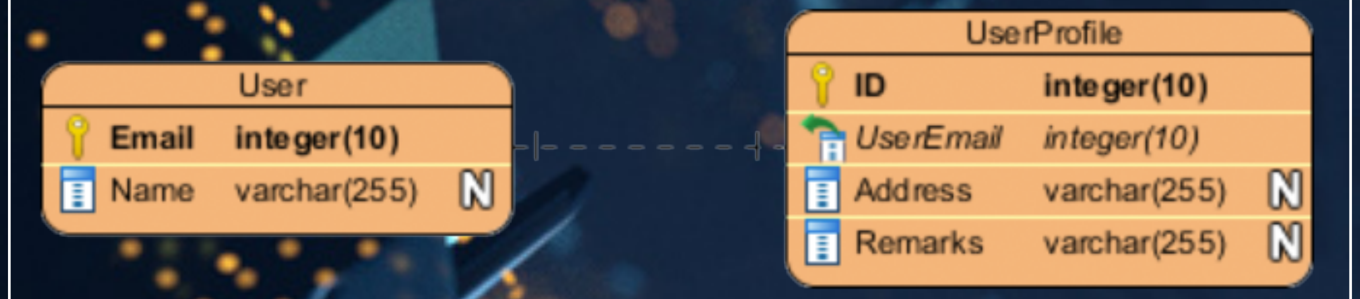
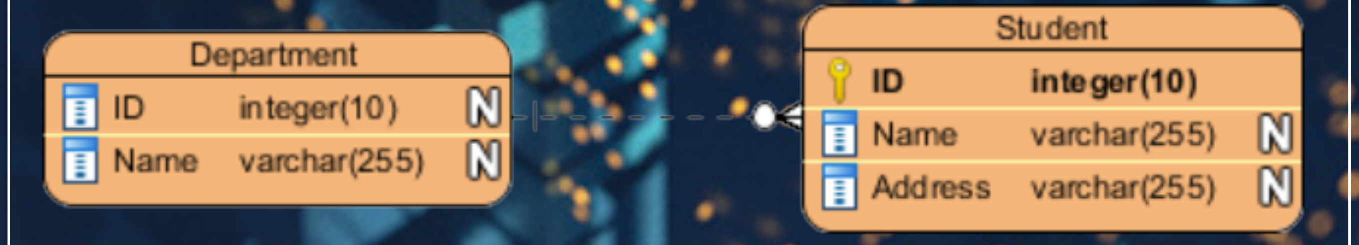
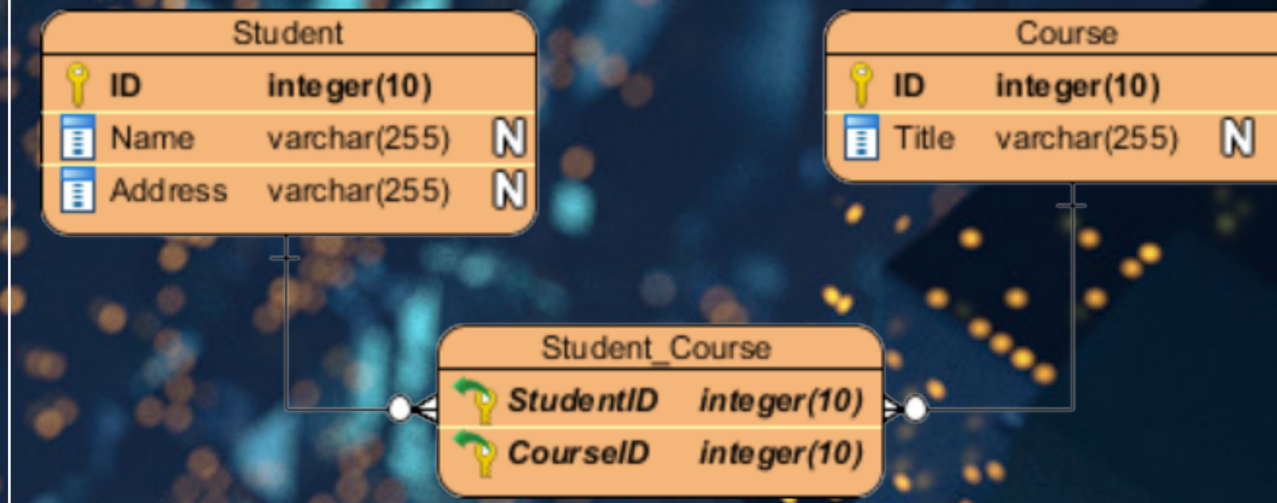
Entity Relationship Diagram (ERD) is a type of structural diagram for use in database design. An ERD contains different symbols and connectors that visualize two important information: **The major entities within the system scope**, and the **inter-relationships among these entities**.

- ***Primary Keys:*** uniquely defines a record in a database table.
  - There must not be two (or more) records that share the same value for the primary key attribute.
  - For example *(ID)*
- ***Foreign Keys:*** reference to a primary key in another table. It is used to identify the relationships between entities.
  - They don't need to be unique
  - For example *(ShipmentID, CourierID)*



SQL - Entity Relationship Diagram

- **Cardinality:** possible number of occurrences in one entity which is associated with the number of occurrences in another.
  - *For example, ONE team has MANY players. Then Team and Player are inter-connected with a one-to-many relationship.*

One to One Relationship	One to Many Relationship	Many to Many Relationship
 User Email integer(10) Name varchar(255) N UserProfile ID integer(10) UserEmail integer(10) Address varchar(255) N Remarks varchar(255) N	 Department ID integer(10) N Name varchar(255) N Student ID integer(10) Name varchar(255) N Address varchar(255) N	 Student ID integer(10) Name varchar(255) N Address varchar(255) N Course ID integer(10) Title varchar(255) N Student_Course StudentID integer(10) CourseID integer(10)

SQL - Statements

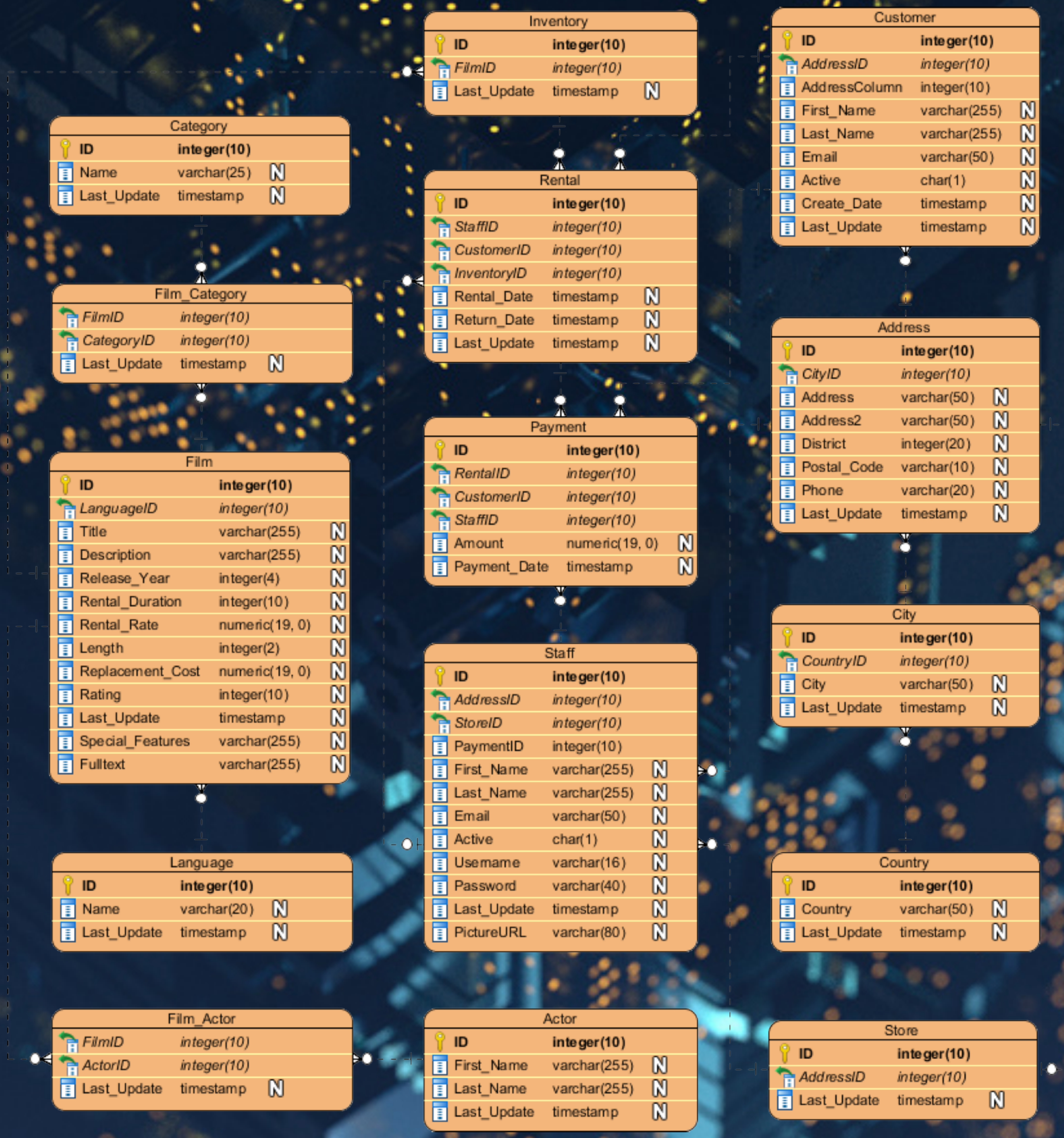
- Databases have four main statements, following the *CRUD (Create, Read, Update, Delete)* paradigm. Statements can essentially be thought of as functions we want to perform over the database. Within a given database, relevant statements could include:
    - **CREATE TABLE**: A statement which creates a new table in the database
    - **SELECT**: Allows you to read and *query* the database to extract records you want
    - **UPDATE**: Allows you to update records within a table
    - **DROP TABLE**: Gives you the power to remove a table from a database
- SQL Statements and Keywords are, by convention, UPPERCASE

SQL - Use Case

It’s time to get our hands dirty. By now you should already have PostgreSQL installed. If not, open the SQL_Setup.md in the repo and follow the steps included in the file.

- We're going to be working with a database known as Pagila
  - PostgreSQL port of an open-source sample database known as Sakila (A sample database for learning purposes)
- Pagila is a database which models a DVD rental store. It features films, actors, film-actor relationships, and a central inventory table that connects films, stores, and rentals.

SQL - Use Case



Inventory

ID	integer(10)	
FilmID	integer(10)	
Last_Update	timestamp	N

Customer

ID	integer(10)	
AddressID	integer(10)	
AddressColumn	integer(10)	
First_Name	varchar(255)	N
Last_Name	varchar(255)	N
Email	varchar(50)	N
Active	char(1)	N
Create_Date	timestamp	N
Last_Update	timestamp	N

Category

ID	integer(10)	
Name	varchar(25)	N
Last_Update	timestamp	N

Rental

ID	integer(10)	
StaffID	integer(10)	
CustomerID	integer(10)	
InventoryID	integer(10)	
Rental_Date	timestamp	N
Return_Date	timestamp	N
Last_Update	timestamp	N

Film_Category

FilmID	integer(10)	
CategoryID	integer(10)	
Last_Update	timestamp	N

Address

ID	integer(10)	
CityID	integer(10)	
Address	varchar(50)	N
Address2	varchar(50)	N
District	integer(20)	N
Postal_Code	varchar(10)	N
Phone	varchar(20)	N
Last_Update	timestamp	N

Payment

ID	integer(10)	
RentalID	integer(10)	
CustomerID	integer(10)	
StaffID	integer(10)	
Amount	numeric(19, 0)	N
Payment_Date	timestamp	N

Film

ID	integer(10)	
LanguageID	integer(10)	
Title	varchar(255)	N
Description	varchar(255)	N
Release_Year	integer(4)	N
Rental_Duration	integer(10)	N
Rental_Rate	numeric(19, 0)	N
Length	integer(2)	N
Replacement_Cost	numeric(19, 0)	N
Rating	integer(10)	N
Last_Update	timestamp	N
Special_Features	varchar(255)	N
Fulltext	varchar(255)	N

City

ID	integer(10)	
CountryID	integer(10)	
City	varchar(50)	N
Last_Update	timestamp	N

Staff

ID	integer(10)	
AddressID	integer(10)	
StoreID	integer(10)	
PaymentID	integer(10)	
First_Name	varchar(255)	N
Last_Name	varchar(255)	N
Email	varchar(50)	N
Active	char(1)	N
Username	varchar(16)	N
Password	varchar(40)	N
Last_Update	timestamp	N
PictureURL	varchar(80)	N

Language

ID	integer(10)	
Name	varchar(20)	N
Last_Update	timestamp	N

Country

ID	integer(10)	
Country	varchar(50)	N
Last_Update	timestamp	N

Film_Actor

FilmID	integer(10)	
ActorID	integer(10)	
Last_Update	timestamp	N

Actor

ID	integer(10)	
First_Name	varchar(255)	N
Last_Name	varchar(255)	N
Last_Update	timestamp	N

Store

ID	integer(10)	
AddressID	integer(10)	
Last_Update	timestamp	N

SQL - SELECT Statement

The most common operation in a database is reading data from it. We can do it using the SELECT keyword.

Let's see the basic anatomy of a SELECT keyword:

- **<u>Operation</u>:** What is going to be done. SELECT followed by the names of columns (* indicates all columns)
- **<u>Data</u>:** FROM where we are getting the data (SELECT * FROM payments: select all columns from the payments entity). Data can also be obtained from the combination of existing columns using arithmetic operations
- **<u>Post-Processing:</u>** It takes the results of a query and sort them or limit them by using ORDER BY and LIMIT
- **Conditional:** Acts as a filter. Usually indicated by WHERE
- **Grouping:** Assemble the rows of a data source using a key created by a GROUP BY clause.

Two small notes:

- *Statements terminate with a semi-colon*
- *SQL ignores whitespace*

SQL - Post Processing

LIMIT

- It allows us to limit the number of records
- It's the last part part of the query

```
SELECT *
FROM payment
LIMIT 10;
```

ORDER BY

- It allows us to specify which columns we want our returned data ordered by
- Ascending by default. We can specify DESC
- We can also sort multiple columns, and the sorting occurs from left to right
- Placed before LIMIT

```
SELECT *
FROM film
ORDER BY rental_rate DESC, length DESC;
LIMIT 10
```

SQL - Practicals (Part I)

Take a look at the ERD in the Pagila database and attempt the first practical (“Use the SELECT query to read the following data from the Pagila database”)

SQL - SELECT Statement

The most common operation in a database is reading data from it. We can do it using the SELECT keyword. Let's see the basic anatomy of a SELECT keyword:

- **Operation:** What is going to be done. SELECT followed by the names of columns (* indicates all columns)
- **Data:** FROM where we are getting the data (SELECT * FROM payments: select all columns from the payments entity). <u>Data can also be obtained from the combination of existing columns using arithmetic operations</u>
- **Post-Processing:** It takes the results of a query and sort them or limit them by using ORDER BY and LIMIT
- **<u>Conditional</u>:** Acts as a filter. Usually indicated by WHERE
- **Grouping:** Assemble the rows of a data source using a key created by a GROUP BY clause.

Two small notes:

- *Statements terminate with a semi-colon*
- *SQL ignores whitespace*

SQL - Conditional

`WHERE:`

- It allows us to specify which columns we want our returned data ordered by
- Common condition symbols are:
  - `>, >=` Greater than, greater than or equal to
  - `<, <=` Less than, less than or equal to
  - `=` Equal to
  - `!=` Not equal to

```
SELECT *
FROM payments
WHERE customer_id = 72;
```

SQL - Arithmetic Operations

- It's possible to return a new column from a combination of existing columns
- We derive/compute this new column using mathematical operators: `+`, `-` , `*` , `/`
- Usually we **alias** the new column using the `AS` keyword

```
SELECT title, (rental_rate/rental_duration) AS rental_rate_per_day
FROM film;
```

SQL - Practicals (Part II)

Take a look at the ERD in the Pagila database and attempt the second practical (“Using Simple Conditional Statements to filter the data”)

SQL - More on Conditional

SQL allows us access to operations that improves the conditional statement `WHERE`

- `LIKE`: Similar to `WHERE =`, but in cases you're not entirely sure what you're searching for
- `IN`: Similar to `WHERE =`, but for more than one condition
- `NOT`: Used to return opposite results from `LIKE` or `IN` (e.g. `NOT LIKE` or `NOT IN`)
- `AND` & `BETWEEN`: Allows us to combine operations
- `OR`: Allows us to find data where one of the conditions we provide to the query is true

SQL - LIKE

- `LIKE` is typically performed on string based columns
- Allows us to match strings using wildcards:
  - One wildcard character in SQL is %
    - Represents zero or more characters
  - Case sensitive
  - Remember to use single quotes in the query

```
SELECT *
FROM film
WHERE description LIKE '%Drama%'
```

SQL - IN

- `IN` can be used with both string and numeric data types
- Essentially allows us to use an =, but over more than one condition

```
SELECT *
FROM language
WHERE name IN ('English', 'Italian')
```

```
SELECT *
FROM film
WHERE language_id IN (1, 2)
```

SQL - NOT

- `NOT` allows us to inverse the results of the previous queries

```
SELECT *
FROM category
WHERE name NOT IN ('Action', 'Animation')

SELECT *
FROM actor
WHERE last_name NOT LIKE '%SON'
```

SQL - AND

- We use AND when we want to run/check against multiple conditions
- We can link as many expressions as we want
  - Of different types too… LIKE, IN, and NOT can all be linked by AND

```
SELECT *
FROM payment
WHERE payment_date >= '2017-01-25'
AND payment_date <= '2017-01-29'

SELECT *
FROM payment
WHERE payment_date < '2017-01-25' AND staff_id = 2
```

SQL - OR

- Similar to `AND`, `OR` combines multiple statements. However, with `OR`, only one of the conditions we specify needs to be true (instead of all cases as is the case with `AND`)

```
SELECT *
FROM film
WHERE rental_duration < 5 OR length > 120
```

SQL - BETWEEN

- When we want to specify a range between data on the same column (like we did in the previous slide), it is easier and cleaner to use BETWEEN
    - Works with dates, strings and numbers

```
SELECT *
FROM payment
WHERE payment_date BETWEEN '2017-01-25' AND '2017-01-29';

SELECT customer_id, payment_date, amount
FROM payment
WHERE amount BETWEEN 10.0 AND 11.99;
```

SQL - Practicals (Part III)

Take a look at the ERD in the Pagila database and attempt the third practical ("Applying filters to SQL queries")