* Relational Databases
  + Tables
    - Rows
    - Columns
* Database Management System (DBMS)
  + Relational DBMS (RDBMS)
* SQL (Structured Query Language)
  + SQL types
    - SQL Server
      * (T-SQL)
      * Transact – SQL)
    - MySQL
    - Oracle SQL
    - PostgreSQL
    - SQLite
  + (Functional) dependency
    - For 2 sets of columns, x & y, x depends on y if, for each possible set of y values there exists exactly 1 possible set of x values
      * If 2 rows have the same y, the values of x must be the same
      * X is a fact of y
  + Candidate key (of a table)
    - Minimal set of columns that every other column depends on
    - The values of any candidate key can uniquely identify that row
  + Primary key (of a table)
    - One of the candidate keys that we chose to actually identify the row
  + Composite key
    - Any key that’s composed of more than one column
  + Foreign key
    - A column that has copies of a table’s primary key to establish a relationship between the tables
    - Referential integrity- every FK value corresponds to an existing row in the other table
* Non-atomic values
  + A value that is more than one unit
  + Problems
    - Difficult to read, search, sort data
* Update anomaly
  + With redundant data, we might now update all copies
* Deletion anomaly
  + If we delete some data, we sometimes lose other data
* Insertion anomaly
  + Becomes impossible to insert some data without some associated data
* Normalization
  + 1NF –
    - No duplicate rows (enforced with primary key)
    - Atomic values (can’t have multiple values in the same column
    - No repeating columns (color1, color2, color3)
  + 2NF –
    - Must meet conditions of 1NF
    - No non-candidate key may depend on part of any composite candidate key
    - Most times will have to split off into new tables to fix
  + 3NF (fully normalized)
    - 2NF
    - No non candidate key columns may depend on any other non candidate key column (non transitive dependencies)
  + Pros and cons of normalization
    - Pros
      * Less redundancy
      * Easier to add to the data model
      * Easier to read and write with consistency
    - Cons
      * Normalization can make some queries slower

Datatypes in SQL

* Numeric
* Integers
  + TINYINT (1 byte) (char/byte in other languages)
  + SMALLINT (2 bytes) (short)
  + INT (4 bytes)
  + BIGINT (8 bytes) (long)
* Floating point
  + FLOAT
  + REAL
  + DECIMAL/NUMERIC (most used)
* Currency
  + MONEY
  + String
* String
  + CHAR/CHARACTER(n)
    - Fixed length string with size n
    - 1 byte per char
  + VARCHAR/CHARACTER VARYING(n)
    - Variable length up to size n
    - 1 byte per char
  + NCHAR(n)
    - Unicode char
  + NVARCHAR(n)
    - Unicode VARCHAR (Primarily used)
* Date/time
  + DATE
    - Dates
  + TIME
    - Times
  + DATETIME
    - Timestamps
    - Not used because of bad precision
  + DATETIME2(n)
    - High precision timestamps
    - Most used

Advanced SELECT statement

* GROUP BY clause
  + By itself it doesn’t do a lot but is useful with aggregate functions (COUNT, AVG, SUM, MIN, MAX)
  + SELECT FirstName, COUNT(FirstName) AS Count FROM SalesLT.Customer GROUP BY FirstName ORDER BY COUNT (FirstName) DESC;
* HAVING does what WHERE does, just using an aggregate function
* Logical order of execution of a SELECT is that the SELECT statement comes toward the end, SELECT runs before ORDER BY
  + FROM
  + WHERE
  + GROUP BY
  + HAVING
  + SELECT
  + ORDER BY