**Databases and RDBMS**

* **\*A database is an organized collection of information**
  + **It imposes rules on the contained data**
  + **Relational storage first proposed by Edgar Codd in 1970**
* **A Relational Data Base Management System provides   
  tools to manage the database**
  + **It parses requests from the user and takes the   
    appropriate action**
  + **The user doesn't have direct access to the stored data**
* Logical Storage
  + Instance
  + Database
  + Schema
  + Table
* Physical Storage
  + Data Files and Log files
  + Data Pages
* The table is the main **building** **block** of any database
* Each **row** is called a **record** or **entity**
* Columns (**fields**) define the **type** of data they contain
* To communicate with the Engine we use **SQL**
  + **Declarative** language
* Logically divided in four sections
  + **Data Definition** – describe the structure of our data
  + **Data Manipulation** – store and retrieve data
  + **Data Control** – define who can access the data
  + **Transaction Control** – bundle operations and allow rollback

**Basic CRUD in SQL Server**

* **\*Structured Query Language**
  + **Declarative language**
  + **Close to regular English**
  + **Supports definition, manipulation and access control of records**
* **Transact-SQL (T-SQL) – SQL Server's version of SQL**
  + **Supports control flow (if statements, loops)**
  + **Designed for writing logic inside the database**
* **Projection**

**Take a subset of the columns**

* **Selection**

**Take a subset of the rows**

* **Join**

**Combine tables by**

**some column**

* **Aliases** rename a table or a column heading
* ***Views*** are **named** (**saved**) **queries**
* **Simplify** complex queries
* **Limit access** to data for certain users
* ***Sequences*** are **special object** in SQL Server
* Similar to **IDENTITY** fields
* Returns an **incrementing value** every time it's used

**Table Relations**

* Steps in the database design process:
* Identify entities
* Identify table columns
* Define a primary key for each table
* Identify and model relationships
* Define other constraints
* Fill tables with test data
* ***Entity tables*** represent objects from the real world

-Most often they are nouns in the specification

* ***Columns*** are clarifications for the entities in the text of the   
  specification
* ***Relationships*** are **dependencies** between the entities

**Database Normalization**

* + - It is a technique of organizing the data in the   
      database
    - **Normalization** is a systematic approach of   
      **decomposing** tables to eliminate data redundancy  
      (repetition) and undesirable characteristics like  
      insertion, update and deletion **anomalies**
    - It is a multi-step process that puts data into **tabular   
      form** removing duplicated data from the relation   
      tables
    - **First Normal Form (1NF)**
    - Table should only have single(atomic) valued attributes/columns
    - Values stored in a column should be of the same domain (same type)
    - All the columns in a table should have unique names
    - The order in which data is stored should not matter
    - **Second Normal Form (2NF)**
    - The table should be in the **First Normal form**
    - It shouldn't have **Partial Dependency** (dependency on part of the primary key)
    - **Third Normal Form (3NF)**
    - The table is in the **Second Normal form**
    - It doesn't have **Transitive Dependency**
* The **foreign** **key** is an **identifier** of a record located in **another** **table**   
  (usually a primary key)
* Using relationships, we **refer** to data instead of **repeating** data
* Country name is **not** **repeated**, it is **referred** to by its **primary** **key**
* **JOIN Statements**
  + - With a **JOIN** statement, we can get data from two tables   
      **simultaneously**
    - **JOIN**s require at least two tables and a "**join condition**"
* **Cascade Operations**
  + - Cascading allows when a change is made to certain   
      entity, this change to apply to all related entities
    - **Cascade** can be either **Delete** or **Update**.
    - Use **Cascade** **Delete** when:
    - The related entities are **meaningless** without the "main" one
    - Do **not** use **Cascade Delete** when:
    - You perform a "**logical** **delete**"
    - Entities are **marked** as deleted (but not actually deleted)
    - In more complicated relations, cascade delete won't work with   
      **circular** references
    - Use **Cascade** **Update** when:
    - The primary key is not identity (not auto-increment) and therefore it **can** be changed
    - Best used with unique constraint
    - Do **not** use **Cascade Update** when:
    - The primary is identity (auto-increment)
    - Cascading can be avoided using **triggers** or **procedures**
* Entity / Relationship Diagrams
  + - **Relational schema** of a DB is the collection of:
    - The schemas of all tables
    - Relationships between the tables
    - Any other database objects (e.g. constraints)
    - The relational schema describes the structure of the database
    - Doesn't contain data, but metadata
    - Relational schemas are graphically displayed in   
      Entity / Relationship diagrams (**E/R Diagrams**)

**Built-in Functions**

* **Aggregate functions**
* **It perform a calculation on a set of values and return a   
  single value**
* **Examples: AVG, COUNT, MIN, MAX, SUM**
* **Analytic functions**
* **It compute an aggregate value based on a group of rows**
* **Unlike aggregate functions, analytic functions can   
  return multiple rows for each group**
* **Ranking functions**
* **Returns a ranking value for each row in a partition**
* **RANK, ROW\_NUMBER, DENSE\_RANK, NTILE (OVER)**
* **Rowset functions**
* **Returns an object that can be used like table references in   
  an statement**
* **OPENDATASOURCE, OPENJSON, OPENXML, OPENROWSET**
* **Scalar functions**
* **Operate on a single value and then return a single value.   
  Scalar functions can be used wherever an expression is valid**

**String Functions**

* + - **Concatenation – combines strings**
    - **CONCAT replaces NULL values with empty string**
    - **CONCAT\_WS combines strings with separator**
    - **SUBSTRING – extracts a part of a string**
    - **LTRIM & RTRIM – remove spaces from either side of string**
    - **LEN – counts the number of characters**
    - **DATALENGTH – gets the number of used bytes**
    - **LEFT & RIGHT – get characters from the beginning or the   
      end of a string**
    - **LOWER & UPPER – change letter casing**
    - **REVERSE – reverses order of all characters in a string**
    - **REPLICATE – repeats a string**
    - **FORMAT – format a value with a valid .NET format string**
    - **CHARINDEX – locates a specific pattern (substring) in a string**
    - **STUFF – inserts a substring at a specific position**

**Math Functions**

* + - **PI – gets the value of Pi as a float (15 –digit precision)**
    - **ABS – absolute value**
    - **SQRT – square root (the result will be float)**
    - **SQUARE – raise to power of two**
    - **POWER – raises value to the desired exponent**
    - **ROUND – obtains the desired precision**
    - **Negative precision rounds characters before the decimal point**
    - **FLOOR & CEILING – return the nearest integer**
    - **SIGN – returns 1, -1 or 0, depending on the value of the sign**
    - **RAND – gets a random float value in the range [0, 1]**
    - **If Seed is not specified, it will be assigned randomly**

**Date Functions**

* + - **DATEPART – extract a segment from a date as an integer**
    - **Part can be any part and format of date or time**
    - **DATEDIFF – finds the difference between two dates**
    - **Part can be any part and format of date or time**
    - **DATENAME – gets a string representation of a date's part**
    - **DATEADD – performs date arithmetic**
    - **Part can be any part and format of date or time**
    - **GETDATE – obtains the current date and time**
    - **EOMONTH – returns the last day of the month**

**Other Functions**

* + - **CAST & CONVERT – conversion between data types**
    - **ISNULL – swaps NULL values with a specified default value**
    - **COALESCE – evaluates the arguments in order and returns the   
      current value of the first expression that initially does not   
      evaluate to NULL**
    - **OFFSET & FETCH – get only specific rows from the result set**
    - **Used in combination with ORDER BY for pagination**
    - **ROW\_NUMBER – always generate unique values without any gaps, even if there are ties**
    - **RANK – can have gaps in its sequence and when values are the same, they get the same rank**
    - **DENSE\_RANK – returns the same rank for ties, but it doesn’t have any gaps in the sequence**
    - **NTILE – Distributes the rows in an ordered partition into a specified number of groups**
* **Wildcards are used with WHERE for partial filtration**
  + - **Similar to Regular Expressions, but less capable**
    - **Example: Find all employees who's first name starts with "Ro"**
    - **Supported characters include:**
    - **ESCAPE – specify a prefix to treat special characters as normal**

**Joins, Subqueries, CTEs**

* **Inner join**
  + - **Join of two tables returning only rows matching the join condition**
* **Left (or right) outer join**
  + - **Returns the results of the inner join as well as unmatched rows from the left (or right) table**
* **Full outer join**
  + - **Returns the results of an inner join along with all unmatched rows**
* **A Cartesian product is formed when:**
* **A join condition is omitted**
* **A join condition is invalid**
* **All rows in the first table are joined to all rows in the second   
  table**
* **To avoid a Cartesian product, always include a valid join   
  condition**