# HOMEWORK - LESSON 2

## Database Systems Overview

1. **What database models do you know?**

I know following database models, depending on their stricture:

* + Hierarchical (tree)
  + Network/graph
  + Relational (table)
  + Object-oriented

1. **Which are the main functions performed by a Relational Database Management System (RDBMS)?**

Relational Database Management Systems (RDBMS) manage data stored in tables as table rows. Single entity spans multiple tables. RDBMS represent a bunch of tables together with the relationships between them. They rely on a strong mathematical foundation: the relational algebra.

RDBMS systems are very mature, rock solid. They typically implement:

* + Creating / altering / deleting tables and relationships between them (database schema)
  + Adding, changing, deleting, searching and retrieving of data stored in the tables
  + Support for the SQL language
  + Transaction management (optional)

1. **Define what is "table" in database terms.**

Database tables consist of data, arranged in rows and columns.

All rows have the same structure; columns have name and type (number, string, date, image, or other), for example:

Table **Workers**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Id** | **FirstName** | **LastName** | **Employer** | **WorkExperience** |
| 1 | Humpty | Dumpty | Telerik | 2 |
| 2 | Baba | Jaga | Telerik | 25 |
| 3 | Princess | Lea | Microsoft | 1 |
| 4 | Jan | Bibiyan | Apple | 10 |

1. **Explain the difference between a primary and a foreign key.**

Primary key is a column of the table that uniquely identifies its rows (usually it is a number). Two records (rows) are different if and only if their primary keys are different.le :

For example, in table **Workers** we can specify the column **Id** as its primary key.

The primary key can be composed by several columns (composite primary key).

The foreign key is an identifier of a record located in another table (usually its primary key), for example - the column **CompanyId** in table **Workers** contains the foreign key values which correspond to the primary key values in the table **Companies** (column **Companies**.**Id**):

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Table **Workers** | | | |  |  | Table **Companies** | |
| **Id** | **FirstName** | **LastName** | **WorkExp** | **CompanyId** |  | **Id** | **Name** |
| 1 | Humpty | Dumpty | Telerik | 1 |  | 1 | Telerik |
| 2 | Baba | Jaga | Microsoft | 3 |  | 2 | BGCompany |
| 3 | Princess | Lea | Telerik | 1 |  | 3 | Microsoft |
| 4 | Jan | Bibiyan | Apple | 5 |  | 4 | SAP |
|  |  |  |  |  |  | 5 | Apple |

By the means of the primary key and foreign key we can create relationships between tables.

The relationships between tables are based on interconnections: primary key / foreign key. By using relationships we avoid repeating data in the database.

In the last example the name of the company is not repeated for each worker (its number is used instead)

1. **Explain the different kinds of relationships between tables in relational databases.**

There are 3 different kinds of relationships between tables in RDBMS:

* + One-to-many – e.g. company/ workers
  + Many-to-many – e.g. worker / profession
  + One-to-one – e.g. example human / worker

Relationship one-to-many (or many-to-one)

A single record in the first table has many corresponding records in the second table: in the example above the record with **Id** = 1 (Telerik) from the table **Companies** has two corresponding records in the table **Workers**: 'Humpty Dumpty' and 'Princess Lea' (and vice-versa).

This type of relationship is used very often.

Relationship many-to-many

Records in the first table have many corresponding records in the second one and vice versa.

Example: the relationship between a table Professions, where different professions are listed with their Id (primary key for Professions), and the table Workers.

The many-to-many relationship is implemented through additional table **WorkersProfessions** which contains two columns - one for the worker **Id** and one for the profession **Id**.

**WorkersProfessions** table has many-to-one relationships with **Workers** and **Professions** tables through their Id columns respectively.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Workers** | | | **WorkersProfessions** | |  | **Professions** | | |
| **Id** | **Name** |  | **WorkerId** | **ProfessionId** |  | | **Id** | **Name** |
| 1 | Peter |  | 4 | 1 |  | | 1 | .NET Ninja |
| 2 | Maria |  | 2 | 3 |  | | 2 | SEO |
| 3 | Nina |  | 2 | 1 |  | | 3 | QA |
| 4 | Ivan |  | 1 | 4 |  | | 4 | Web Dev |
|  |  |  | 3 | 2 |  | | 5 | Front End |
|  |  |  | 4 | 2 |  | |  |  |

Both columns in table **WorkersProfessions** have repeated values while the corresponding **Id** columns in tables **Workers** and **Professions** are defined as primary keys and contain unique values.

Relationship one-to-one

A single record in a table corresponds to a single record in the other table. The "one-to-one" relashionship is used to model inheritance between tables.

Example:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | |  | | | | **Managers** | |
| **Persons** | |  |  |  |  | **Id** | **Title** |
| **Id** | **Name** | **Education** |  |  |  | 1 | The Boss |
| 1 | Peter | Ph.D. in CS |  |  |  |  |  |
| 2 | Ivan | Engineer |  |  |  | **Workers** | |
| 3 | Steve | Tech School |  |  |  | **Id** | **Position** |
| 4 | Maria | Tech School |  |  |  | 2 | Ch.Expert |
| 5 | Nina | College |  |  |  | 3 | Expert |
| 6 | Mimi | HighSchool |  |  |  | 4 | Expert |
|  | | |  |  |  | 5 | Administrator |
| **Id** column in **Persons** table is a primary key | | | | | | 6 | Secretary |

The Id columns in tables **Managers** and **Workers** are used as primary and foreign keys.

1. **When is a certain database schema normalized? What are the advantages of normalized databases?**

The database schema is normalized when it contains much of repeated data.

Normalization of the relational database schema removes duplicated data and avoids repetitions.

Non-normalized schemas can contain many data repetitions.

There are several different levels (normal forms) for RDBMS normalization.

* + 1-st Normal Form (1NF) characteristics

Data is stored in tables

Fields in the rows are atomic (inseparable) values

There are no repetitions within a single row

A primary key is defined for each table

Example (**ISBN** column is the primary key):

|  |  |  |  |
| --- | --- | --- | --- |
| **BookTitle** | **ISBN(PK)** | **Author** | **AuthorEmail** |
| **The Forsyte Saga** | 3847028437 | John Galsworthy | jgalsworthy@abv.bg |
| Les Trois Mousquetaires | 7234534450 | Alexandre Dumas | adumas@abv.bg |

* + 2-nd Normal Form (2NF) characteristics

Retains all requirements of 1-st Normal Form

There are no columns that do not depend on part of the primary key (if it consists of several columns)

Example:

|  |  |  |  |
| --- | --- | --- | --- |
| **BookTitle(PK)** | **Author(PK)** | **Price** | **AuthorEmail** |
| **The Forsyte Saga** | John Galsworthy | 37.25 | jgalsworthy@abv.bg |
| Les Trois Mousquetaires | Alexandre Dumas | 19.95 | adumas@abv.bg |

In this example the **BookTitle** and **Author** columns form a **composite** primary key; the rest of columns depend on specific part of this composite primary key as follows:

- **Price** depends on the **BookTitle**

**- AuthorEmail** depends on the **Author**

* + 3-rd Normal Form (3NF) characteristics

Retains all requirements of 2-nd Normal Form

The only dependencies between columns are of type "a column depends on the PK". i.e., there are no transitive functional dependencies. By transitive functional dependency, we mean we have the following relationships in the table: A is functionally dependent on B, and B is functionally dependent on C. In this case, C is transitively dependent on A via B.

Example: Table **BOOKS**

|  |  |  |  |
| --- | --- | --- | --- |
| **BookID** | **GenreID** | **GenreType** | **Price** |
| 1 | 1 | Gardening | 5.80 |
| 2 | 2 | Sports | 12.30 |
| 3 | 1 | Gardening | 15.00 |
| 4 | 3 | Travel | 25.34 |
| 5 | 2 | Sports | 9.65 |

In this table, BookID determines GenreID, and GenreID determines GenreType. Therefore, BookID determines GenreType via GenreID and we have transitive functional dependency, and this structure does not satisfy third normal form.

To bring this table to third normal form, we split the table into two as follows:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **BOOKS** | | |  | **GENRES** | |
| **BookID** | **GenreID** | **Price** |  | **GenreID** | **GenreType** |
| 1 | 1 | 5.80 |  | 1 | Gardening |
| 2 | 2 | 12.30 |  | 2 | Sports |
| 3 | 1 | 15.00 |  | 3 | Travel |
| 4 | 3 | 25.34 |  |  |  |
| 5 | 2 | 9.65 |  |  |  |

Now all non-key attributes are fully functional dependent only on the primary key. In the table **BOOKS**, both **GenreID** and **Price** are only dependent on **BookID**. In **GENRE**, **GenreType** is only dependent on **GenreID**.

* + 4-th Normal Form (4NF) characteristics

Retains all requirements of 3-rd Normal Form

There is one column at most in each table that can have many possible values for a single key (multi-valued attribute).

Example: Suppose that employees can be assigned to multiple projects. The employees can have multiple job skills as shown in database as shown in the table **EMP\_PROJECT\_SKILL**.

|  |  |  |
| --- | --- | --- |
| **EMP\_PROJECT\_SKILL** |  |  |
| **EmpID** | **ProjectNo** | **Skill** |
| 2546 | 7 | Web Design |
|  | 8 | SEO |
|  |  | QA |
| 2547 | 3 | Front-end |
|  | 4 | Java Developer |

In order to normalize it we flatten the database with first normal form:

|  |  |  |
| --- | --- | --- |
| **EMP\_PROJECT\_SKILL** |  |  |
| **EmpID** | **ProjectNo** | **Skill** |
| 2546 | 7 | Web Design |
| 2546 | 7 | SEO |
| 2546 | 7 | QA |
| 2546 | 8 | Web Design |
| 2546 | 8 | SEO |
| 2546 | 8 | QA |
| 2547 | 3 | Front-end |
| 2547 | 3 | Java Developer |
| 2547 | 4 | Front-end |
| 2547 | 4 | Java Developer |

This database shows that **ProjectNo** and **Skill** are independent multi-valued columns about **EmpNo** that is it contains "a multi-valued dependency" (MVD).

This form of the table is full of redundancy and needs to be re-designed. Since the database contains MVDs, so it should be decomposed with the help of rule of fourth normal form. The database contain the following MVDs:

* + **EmpNo** --> --> **ProjectNo**
  + **EmpNo**--> --> **Skill**

Here, **ProjectNo** and **Skill** are independent to each other, so it should be decomposed in to following database according to forth normal form.

* + **EMP\_PROJECT** (**EmpNo**, **ProjectNo**)
  + **EMP\_SKILL** (**EmpNo**, **Skill**)

|  |  |
| --- | --- |
| **EMP\_PROJECT** |  |
| **EmpID** | **ProjectNo** |
| 2546 | 7 |
| 2546 | 8 |
| 2547 | 3 |
| 2547 | 4 |
| **EMP\_SKILL** |  |
| **EmpID** | **Skill** |
| 2546 | Web Design |
| 2546 | SEO |
| 2546 | QA |
| 2547 | Front-end |
| 2547 | Java Developer |

1. **What are database integrity constraints and when are they used?**

Database integrity constraints ensure data integrity in the database tables. They enforce data rules which cannot be violated:

* + Primary key constraint

Ensures that the primary key of a table has unique value for each table row

* + Unique key constraint

Ensures that all values in a certain column (or a group of columns) are unique

* + Foreign key constraint

Ensures that the value in given column is a key from another table

* + Check constraint

Ensures that values in a certain column meet some predefined condition.

1. **Point out the pros and cons of using indexes in a database.**

Indices speed up searching of values in a certain column or group of columns. They are usually implemented as B-trees.

Indices can be built-in the table (clustered) or stored externally (non-clustered).

Since adding and deleting records in indexed tables is slower, indices should be used for big tables only (e.g. 50 000 rows).

1. **What's the main purpose of the SQL language?**

The Structured Query Language is a standardized declarative language for manipulation of relational databases.

SQL-99 is currently in use in most databases, but SQL:201 is the last revision.

SQL language supports:

* + Creating, altering, deleting tables and other objects in the database
  + Searching, retrieving, inserting, modifying and deleting table data (rows)

SQL consists of:

* + DDL – Data Definition Language

CREATE, ALTER, DROP commands

* + DML – Data Manipulation Language

SELECT, INSERT, UPDATE, DELETE commands

Most major RDBMs support stored procedures (database-level procedures) - a batch of SQL statements that has been created and stored in the database. Stored procedure will accept input parameters so that a single procedure can be used over the network by several clients using different input data. Stored procedure will reduce network traffic and increase the performance. If we modify stored procedure all the clients will get the updated stored procedure.

Stored procedures consist of SQL-like code stored in the database. This code is executed inside the database server and performs much faster than an external code. Can accept parameters and return results (a single value or record sets). Data is locally accessible.

Stored procedures are written in a language extension of SQL:

* + T-SQL – in Microsoft SQL Server
  + PL/SQL – in Oracle

Another feature of DBMS are views. Views are named SQL SELECT queries which are used as tables. They simplify data access, facilitate writing of complex SQL queries, and are used also to apply security restrictions: e.g. a certain user isn't given permissions on any of the tables in the database - the user is given permissions on few views (subset of DB) and few stored procedures only.

Triggers are special stored procedures that are activated when some event occurs, for instance:

* + When inserting a record
  + When changing a record
  + When deleting a record

Triggers can perform additional data processing of the affected rows, e.g.

* + To change the newly added data
  + To maintain logs and history

1. **What are transactions used for? Give an example.**

Transactions are a sequence of database operations which are executed as a single unit:

* + Either all of them execute successfully
  + Or none of them is executed at all

Example:

A bank transfer from one account into another (withdrawal + deposit)

If either the withdrawal or the deposit fails the entire operation should be cancelled

Illustration of DB Transactions Lifecycle:

Transactions guarantee the consistency and the integrity of the database. All changes in a transaction are temporary. Changes become final when COMMIT is successfully executed.

At any time all changes done in the transaction can be canceled by executing ROLLBACK.

All operations are executed as a single unit: either all of them pass or none of them.

1. **What is a NoSQL database?**

A NoSQL database provides a mechanism for [storage](https://en.wikipedia.org/wiki/Computer_data_storage) and [retrieval](https://en.wikipedia.org/wiki/Data_retrieval) of data that is modeled in means other than the tabular relations used in [relational databases](https://en.wikipedia.org/wiki/Relational_database):

* + Data stored as documents
  + Single entity (document) is a single record
  + Documents do not have a fixed structure

NoSQL (non-relational) databases use non-relational model and schema-free document storage. They still support:

* + CRUD operations (create, read, update, delete)
  + Indexing and querying
  + Concurrency and transactions

Key benefits of NoSQL DBMSs are:

* + Highly optimized for append / retrieve
  + Great performance and scalability

Given the above, a question arises: NoSQL == "No SQL" or "Not Only SQL"?

1. **Explain the classical non-relational data models.**

Classical non-relational data models:

* + Document model

The central concept of a document store is the notion of a "document". While each document-oriented database implementation differs on the details of this definition, in general, they all assume that documents encapsulate and encode data (or information) in some standard formats or encodings (e.g. JSON strings). Documents are addressed in the database via a unique key that represents that document. One of the other defining characteristics of a document-oriented database is that in addition to the key lookup performed by a key-value store, the database offers an API or query language that retrieves documents based on their contents

Different implementations offer different ways of organizing and/or grouping documents: collections, tags, non-visible metadata, and directory hierarchies.

Compared to relational databases, for example, collections could be considered analogous to tables and documents analogous to records. But they are different: every record in a table has the same sequence of fields, while documents in a collection may have fields that are completely different.

* + Key-value model

Key-value (KV) stores use the [associative array](https://en.wikipedia.org/wiki/Associative_array) (also known as a map or dictionary) as the ir fundamental data model. In this model, data is represented as a collection of key-value pairs, such that each possible key appears at most once in the collection.

The key-value model is one of the simplest non-trivial data models, and richer data models are often implemented on top of it. The key-value model can be extended to an ordered model that maintains keys in [lexicographic order](https://en.wikipedia.org/wiki/Lexicographical_order). This extension is powerful, in that it can efficiently process key ranges. Key-value stores can use [consistency models](https://en.wikipedia.org/wiki/Consistency_model) ranging from [eventual consistency](https://en.wikipedia.org/wiki/Eventual_consistency) to [serializability](https://en.wikipedia.org/wiki/Serializability). Some support ordering of keys. Some maintain data in memory (RAM), while others employ [solid-state drives](https://en.wikipedia.org/wiki/Solid-state_drive) or [rotating disks](https://en.wikipedia.org/wiki/Hard_disk_drive).

* + Hierarchical key-value model

This type of NoSQL DBMS is based on hierarchy of key-value pairs to store large amount of data in a tree. Suitable for **use** in highly concurrent environment, allows for organizing and querying large amount of data efficiently. This model is transactional, highly concurrent, with an easy-to-use API, with JSON import/export and query support provided.

* + Wide-column model

The Wide Column model consists of a key on which data can be queried. Each key holds a ‘single’ value that can have a variable number of columns (key-value model with schema). Each column can nest other columns. Columns can be grouped into a family and each column can be part of multiple column families. Like the Object model the schema of a Wide Column store is flexible. It can be highly performing and It cannot be used for each kind of application. Optimized for queries over large datasets, and store columns of data together, instead of rows.

* + Object model - a set of OOP-style objects

Object oriented DBMSs (OODBMS) has become the preferred technique used by developers (largely due to its superior efficiency and flexibility).

OODBMS should be used when there is a business need, high performance required, and complex data is being used. Due to the object oriented nature of the database model, it is much simpler to approach a problem with these needs in terms of objects. The result can be a performance increase of ten to one thousand times while writing as little as 40% of the code (this is because it requires no intermediate language such as SQL; everything is programmed in the OO language of choice). This code can be directly applied to a database, and thus saves time and money in development and maintenance.

1. **Give few examples of NoSQL databases and their pros and cons.**

Most widely used NoSql databases:

* + Redis - ultra-fast in-memory data structures server;
  + MongoDB - Mature and powerful JSON-document database
  + CouchDB - JSON-based document database with REST API
  + Cassandra - an [open source](https://en.wikipedia.org/wiki/Open_source_software) [distributed](https://en.wikipedia.org/wiki/Distributed_database) [database management system](https://en.wikipedia.org/wiki/Database_management_system)

Cassandra was designed to handle large amounts of data across many commodity servers, providing high availability with no single point of failure. Cassandra offers robust support for clusters spanning multiple datacenters, with asynchronous masterless replication allowing low latency operations for all clients.

Cassandra also places a high value on performance. Cassandra's data model is a partitioned row store with tunable consistency. Rows are organized into [tables](https://en.wikipedia.org/wiki/Table_%28database%29); the first component of a table's primary key is the partition key; within a partition, rows are [clustered](https://en.wikipedia.org/wiki/Clustered_index) by the remaining columns of the key. Other columns may be indexed separately from the primary key.

Tables may be created, dropped, and altered at runtime without blocking updates and queries.

Cassandra does not support [joins](https://en.wikipedia.org/wiki/Join_%28SQL%29) or [subqueries](https://en.wikipedia.org/wiki/Correlated_subquery). Rather, Cassandra emphasizes [de-normalization](https://en.wikipedia.org/wiki/Denormalization) through features like collections.

1. **Distributed wide-column database**
2. **References:**
   * "Database Systems Overview" - presentation, "Databases" course, "Web development" track - Telerik Software Academy 2015;
   * http://www.1keydata.com/database-normalization/third-normal-form-3nf.php;
   * http://ecomputernotes.com/database-system/rdbms/fourth-normal-form;
   * [http://en.wikipedia.org/wiki/SQL#Standardization](http://en.wikipedia.org/wiki/SQL)
   * https://en.wikipedia.org/wiki/NoSQL
   * http://www.codeproject.com/Articles/838398/A-first-look-at-NoSQL-and-MongoDB-in-particular
   * http://db-engines.com/en/ranking