# Databases Overview Homework

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

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

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

* Manage data stored in tables
* 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).

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 its is a number)
* The foreign key is an identifier of a record located in another table (usually its primary key)

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

* Three kinds of relationships:
  + One-to-many
    - A single record in the first table has many corresponding records in the second table.
  + Many-to-many
    - Records in the first table have many correspon-ding records in the second one and vice versa. It is implemented through additional table.
  + One-to-one
    - A single record in a table corresponds to a single record in the other table. Used to model inheritance between tables.

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

* Normalization of the relational schema removes repeating data
* 1-st Normal Form
  + 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
* 2-nd Normal Form
  + 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)
* 3-rd Normal Form
  + Retains all requirements of 2-nd Normal Form
  + The only dependencies between columns are of type "a column depends on the PK"
* 4-th Normal Form
  + 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)

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

* **Integrity constraints ensure data integrity in the database tables**
* **Enforce data rules which cannot be violated**

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

* Pros:
  + Indices speed up searching of values in a certain column or group of columns
  + Indices can be built-in the table (clustered) or stored externally (non-clustered)
* Cons:
  + 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?**

* Standardized declarative language for manipulation of relational databases

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

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

* NoSQL (non-relational) databases
  + Use document-based model (non-relational)
  + Schema-free document storage
    - Still support CRUD operations (create, read, update, delete)
    - Still support indexing and querying
    - Still supports concurrency and transactions
  + Highly optimized for append / retrieve
  + Great performance and scalability

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

* Document model
  + Set of documents, e.g. JSON strings
* Key-value model
  + Set of key-value pairs
* Hierarchical key-value
  + Hierarchy of key-value pairs
* Wide-column model
  + Key-value model with schema
* Object model
  + Set of OOP-style objects

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

* Redis
  + Pros:
    - Powerful data types and powerful commands to leverage them. Hashes, Sorted Sets, Lists, and more.
    - Persistence to disk, by default.
    - Transactions with optimistic locking (WATCH/MULTI/EXEC)
    - Pub/sub. Extremely fast.
    - Values up to 512MB in size (memcached limited to 1MB per key)
    - Lua scripting (as of 2.6)
    - Extremely fast at everything. Benchmarks are often conflicting, but this much is clear: when used like memcached Redis falls somewhere between nearly as fast or maybe even a little faster. Like memcached it is often bound by available network or even memory bandwidth instead of CPU or other bottlenecks and will rarely be the culprit when your app is slowing down.
  + Cons:
    - There are no access controls; it's not possible to configure a Redis server, a db that it's serving, nor any sort of key glob/pattern as being read-only to a subset of your clients
* MongoDB
  + Pros:
    - schema-less. If you have a flexible schema, this is ideal for a document store like MongoDB. This is difficult to implement in a performant manner in RDBMS
    - ease of scale-out. Scale reads by using replica sets. Scale writes by using sharding (auto balancing). Just fire up another machine and away you go. Adding more machines = adding more RAM over which to distribute your working set.
    - cost. Depends on which RDBMS of course, but MongoDB is free and can run on Linux, ideal for running on cheaper commodity kit.
    - you can choose what level of consistency you want depending on the value of the data (e.g. faster performance = fire and forget inserts to MongoDB, slower performance = wait til insert has been replicated to multiple nodes before returning)
  + Cons:
    - Data size in MongoDB is typically higher due to e.g. each document has field names stored it
    - less flexibity with querying (e.g. no JOINs)
    - no support for transactions - certain atomic operations are supported, at a single document level
    - at the moment Map/Reduce (e.g. to do aggregations/data analysis) is OK, but not blisteringly fast. So if that's required, something like Hadoop may need to be added into the mix
    - less up to date information available/fast evolving product
* CouchDB
  + Pros:
    - CouchDB is somewhat faster on read, at least until it reaches about 1000 queries.
    - You can easily change the schema of your data.
    - CouchDB is highly scalable because of it’s “shared nothing” clustering. This allows for individual server failures with zero downtime in larger environements.
    - You can easily retrieve and manipulate data with client-side languages like jQuery, however this should be handled with care!
  + Cons:
    - CouchDB performs a much slower write, especially when thePUTting a lot of data for one document.
    - You can’t really perform complex dynamic queries, which results in having to perform joins and sorting logic yourself.
    - We have 30 years of knowledge about relational databases, NoSQL solutions like CouchDB aren’t that mature yet and could be risky to adopt.
* Cassandra
  + Pros:
    - Open Source
    - Peer to Peer Architecture
    - Elastic Scalability
    - High Availability and Fault Tolerance
    - Column Oriented
    - Schema-Free
  + Cons:
    - Sub-millisecond consistency
    - Transactions (Local or XA)
    - Ad-hoc querying (In Cassandra, you need to model your data model based on the query patterns)
    - Consistent indexes on values (I know Cassandra supports indexes, but because of lack of transactions it's possible to have indexes that are out-of-sync with the data)
    - And to less extent, consistent read/write latency of heavy load (this is mainly because of Java's Garbage Collection)