4. Architektury IoT systémů, databáze

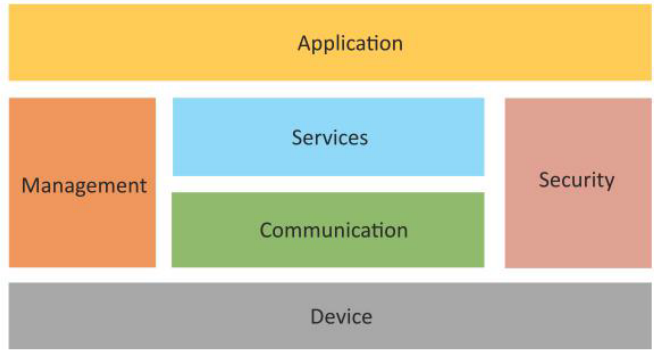
B0B37NSI – Návrh systémů IoT

Stanislav Vítek

Katedra radioelektroniky
Fakulta elektrotechnická
České vysoké učení v Praze

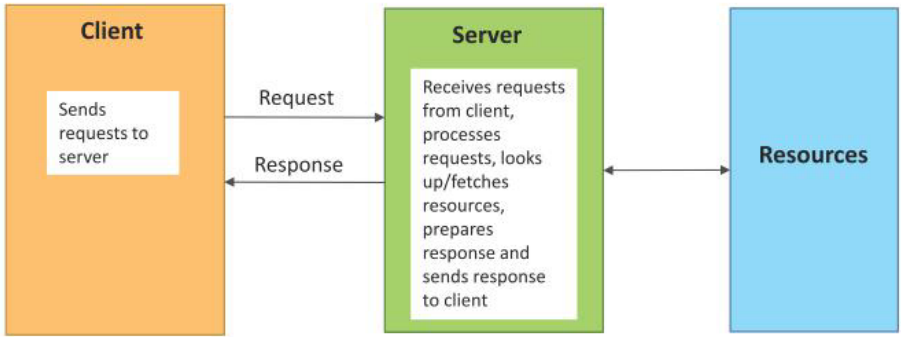
Logický design IoT systému

- Logický návrh systému IoT se týká abstraktního reprezentace entit a procesů, bez ohledu na nízkoúrovňová specifika implementace.
- Systém IoT se skládá z řady funkčních bloků které systému poskytují možnosti identifikace, snímání, ovládání, komunikaci a řízení.



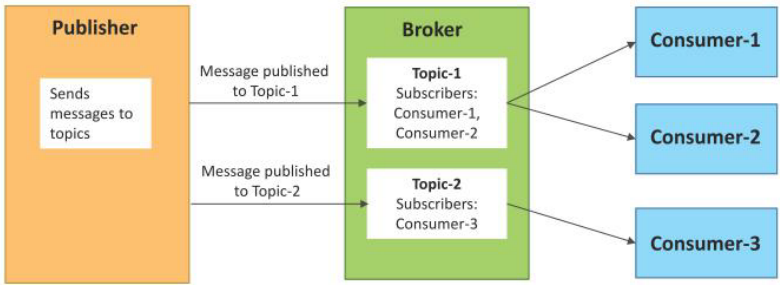
Komunikační model Request-Response

- Request-Response je komunikační model, ve kterém klient posílá požadavky server na tyto požadavky odpovídá.
- Když server obdrží požadavek, rozhodne se, jak odpoví, načte data, načte zdroj reprezentace (např. šablona odpovědi), připraví odpověď a poté odešle odpověď klientovi.



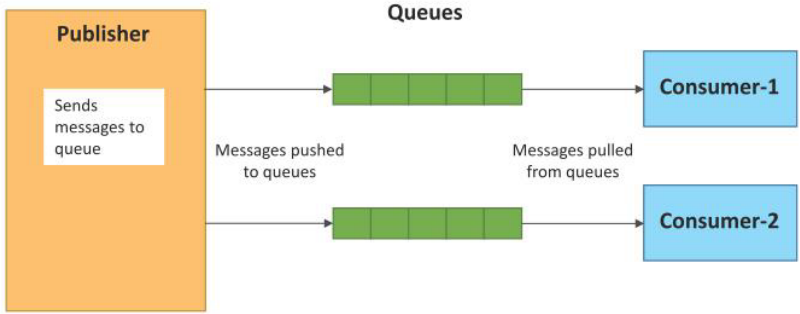
Komunikační model Publish-Subscribe

- Publish-Subscribe je model, který zahrnuje vydavatele (publish), zprostředkovatele (broker) a spotřebitele (consumer) informace.
- Zdrojem dat je vydavatel, který posílá data v rámci téma (topic), které spravuje zprostředkovatel. Vydavatel neví nic o spotřebiteli.
- Spotřebitel se příhlašuje k odběru (subscribe) téma
- Zprostředkoval přeposílá data od vydavatele všem přihlášeným spotřebitelům



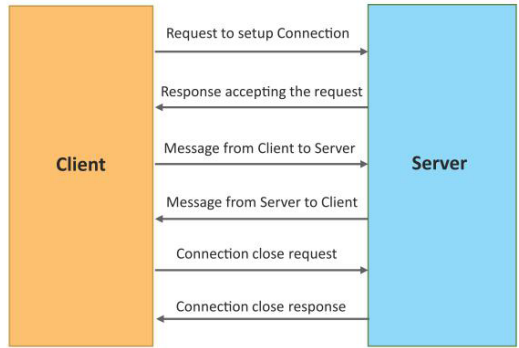
Komunikační model Push-Pull

- Push-Pull je komunikační model, ve kterém producenti dat odesílají data do front a konzumenti je z front odebírají. Producenti nemusí vědět o konzumentech.
- Fronty pomáhají oddělit zasílání zpráv mezi producenty a spotřebiteli.
- Fronty také fungují jako vyrovnávací paměť, která pomáhá v situacích, kdy dochází k výpadkům z důvodu nesouladu mezi rychlostí, kterou producenti odesílají data, a rychlostí, kterou spotřebitelé data stahují.



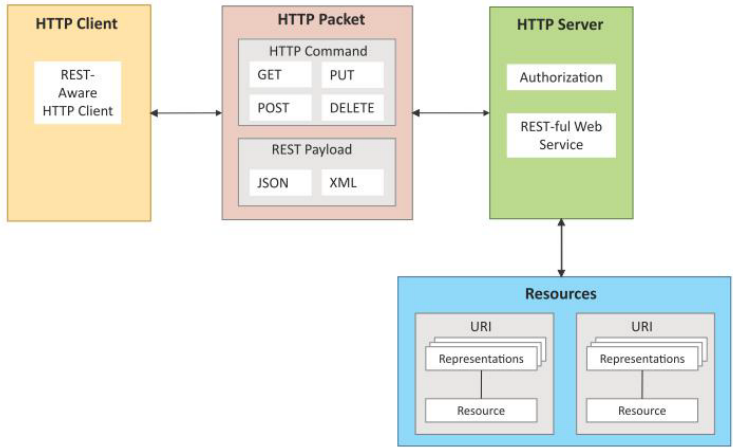
Komunikační model Exclusive Pair

- Exclusive Pair je obousměrný, plně duplexní komunikační model, který používá trvalé spojení mezi klientem a serverem.
- Jakmile je spojení nastaveno zůstane otevřené, dokud klient odešle požadavek na ukončení.
- Klient a server mohou posílat si navzájem posílat zprávy po navázání spojení.

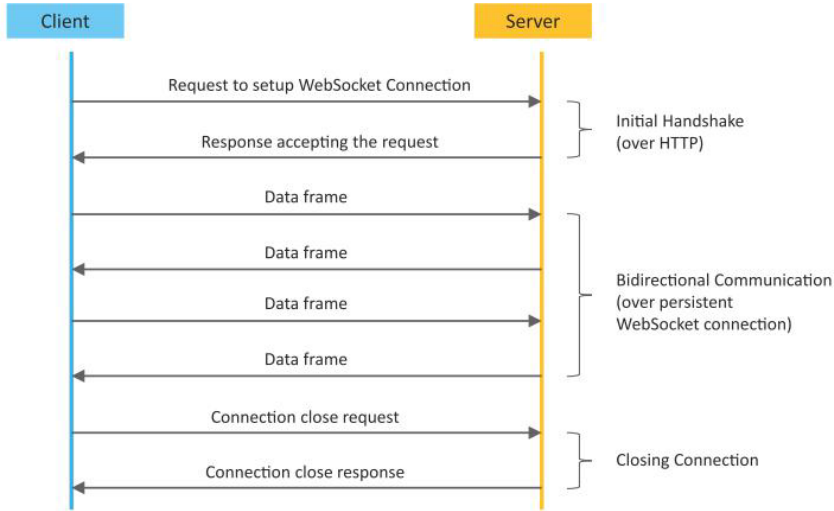


Komunikační rozhraní REST

- Representational State Transfer (REST) je soubor principů pro návrh webových služeb a jejich rozhraní (API)
- REST je zaměřen na zdroje systému (resources) a na to, jak jsou stavy zdrojů adresovány a přenášeny.
- Rozhraní API REST se řídí principy modelu Request-Response.



Komunikační rozhraní WebSocket

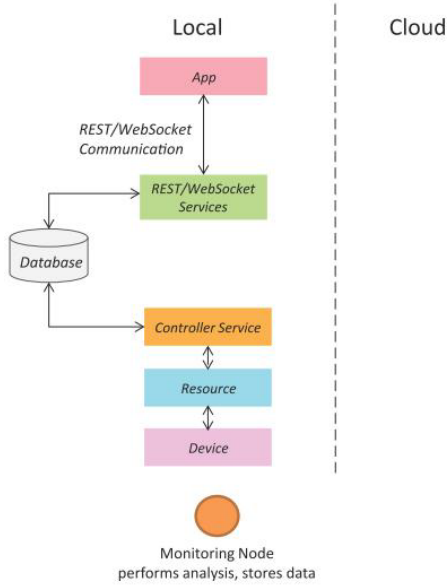


Komponenty IoT systémů

- Zařízení: umožňuje identifikaci, vzdálené snímání, ovládání a vzdálené monitorování.
- Zdroje: softwarové komponenty v zařízení IoT pro přístup k informacím ze senzorů, jejich zpracování a ukládání nebo pro ovládání akčních členů připojených k zařízení. Zdroje zahrnují také softwarové komponenty, které umožňují přístup zařízení k síti.
- Služba řídicí jednotky: nativní služba, která běží v zařízení a komunikuje s webovými službami – odesílá data ze zařízení do webové služby a přijímá příkazy z aplikace (prostřednictvím webových služeb) pro ovládání zařízení.
- Databáze: lokální, nebo v cloudu, ukládá data generovaná zařízením IoT.
- Webová služba: propojení mezi zařízeními IoT, aplikací, databází a analytickými komponentami, typicky implementována pomocí protokolu HTTP a principů REST.
- Analytická složka: analýza a generování výsledků ve formě, která je pro uživatele srozumitelná a snadno dostupná.
- Aplikace: poskytují rozhraní, které mohou uživatelé používat k ovládání a sledování IoT systému a prohlížení zpracovaných dat.

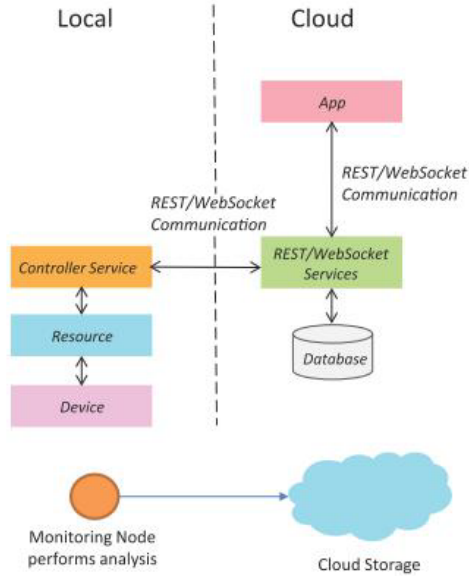
IoT – úroveň 1

- IoT systém úrovně 1 má jeden uzel/zařízení, který provádí snímání a/nebo provádí činnost pomocí aktuátoru a ukládá data, provádí analýzu a je hostitelem aplikaci
- IoT systémy úrovně 1 jsou vhodné pro modelování nízkonákladových řešení s nízkou složitostí, kde jsou data nejsou nijak velká a požadavky na analýzu nejsou výpočetně náročné.



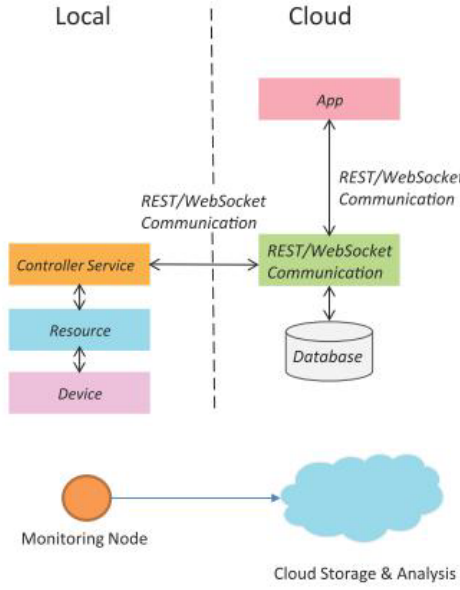
IoT – úroveň 2

- IoT systém úrovně 2 má jeden uzel, který provádí snímání a/nebo ovládání a lokální analýzu.
- Data jsou uložena v cloudu a aplikace je obvykle založena na cloudu.
- Vhodné pro řešení, kde se jedná o velký objem dat, nicméně primární analýza není výpočetně náročná a lze ji provést lokálně.



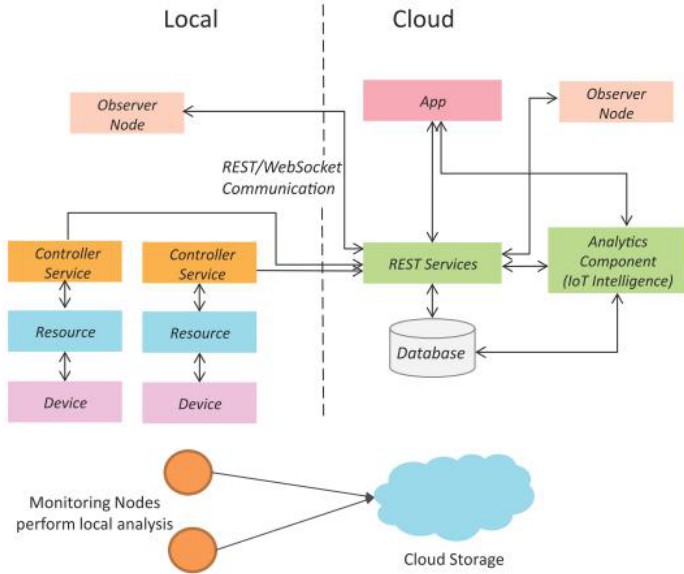
IoT – úroveň 3

- IoT systém úrovně 3 má jeden uzel. Data jsou uložena a analyzována v cloudu a aplikace je založena na cloudu.
- Vhodné pro řešení v aplikacích, kde senzor produkuje velká data a analýza je výpočetně náročná.



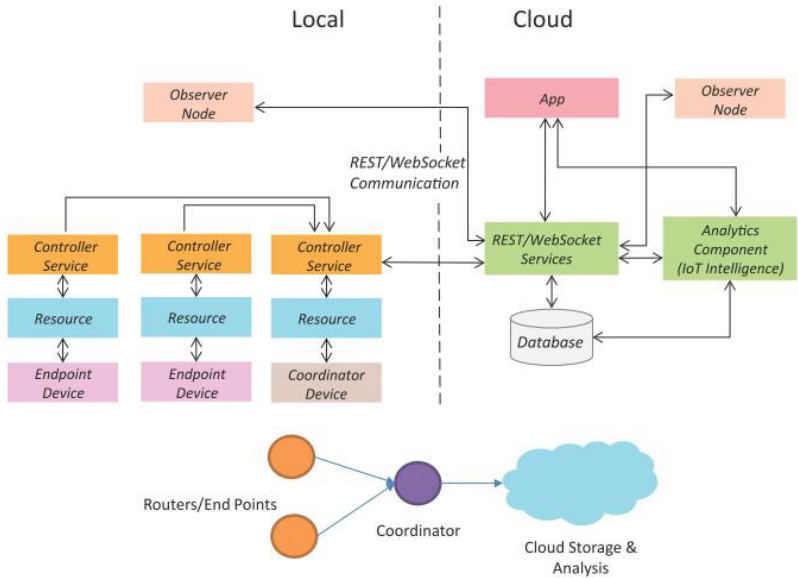
IoT – úroveň 4

- Systém IoT 4. úrovně má více uzlů, které provádějí místní analýzu. Data jsou uložena v cloudu a aplikace je založena na cloudu.
- Úroveň 4 obsahuje místní a cloudové pozorovací uzly, které mohou přihlásit k odběru a přijímat informace shromážděné v cloudu ze zařízení internetu věcí.
- Vhodné pro řešení, kde se používá více uzlů, přičemž data jsou velká a analýza je výpočetně náročná.



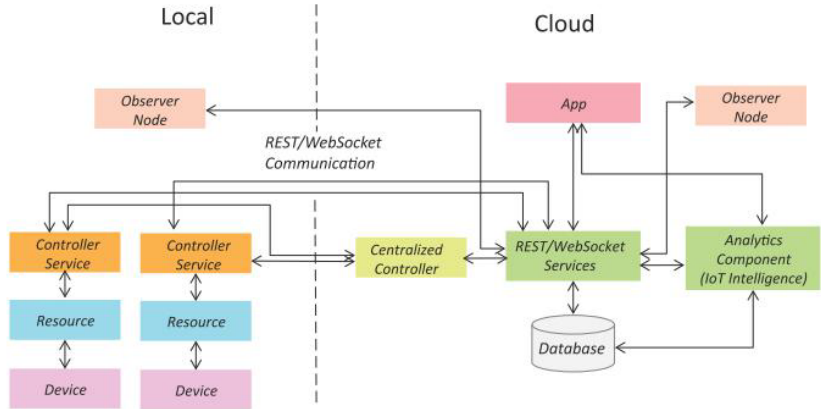
IoT – úroveň 5

- IoT systém 5. úrovně má více koncových zařízení a jeden koordinační uzel.
- Koordinační uzel shromažďuje data z koncových zařízení a odesílá je do cloudu.
- Data se ukládají a analyzují v cloudu
- Typicky bezdrátové senzorové sítě, kde jsou velká data a analýza je výpočetně náročná.



IoT – úroveň 6

- IoT systém 6. úrovně má více nezávislých koncových uzlů, které odesílají data do cloudu
- Analytická komponenta ukládá výsledky do cloudové databáze.
- Výsledky jsou vizualizovány pomocí cloudové aplikace.
- Centralizovaná řídicí jednotka zná stav koncových uzlů a posílá řídicí příkazy.



Databáze

- **Databáze** je místo, kam lze ukládat data a definovat operace, které s nimi lze provádět.
- Zdroje dat vhodné k ukládání
  - Logy webových/emailových serverů, routerů, ...
  - Sítě senzorů
  - Internet věcí
  - Stroje ovládané počítači

  Např. jeden zámořský let Boeingu 777 vygeneruje odhadem 640TB dat
- Relační databázové systémy
  - Založeny na relačním modelu dat a relační algebře
  - Pro manipulaci s daty se nejčastěji využívá jazyk SQL (Structured Query Language)
  - Zaručená konzistence dat
- NoSQL
  - Zaměření na výkon a rychlost zpracování
  - Vysoká škálovatelnost, schéma spravovaných dat se může měnit
  - Připouští dočasnou nekonzistenci

Relační databáze

- Základem relačních databází jsou databázové **tabulky**, které jsou na sebe určitým způsobem závislé – existuje mezi nimi jistá logická vazba (**relace**).
  - Poprvé je definoval v roce 1970 Edgar Codd (IBM Research Lab)
- Tabulky jsou též nazývány entitami – jsou chápány jako prvek reálného světa.
- Každá tabulka je tvořena **sloupci** a **řádky**, sloupce reprezentují vlastnosti této entity.
  - Každý sloupec musí mít jedinečný název a určený datový typ podle toho, jaká data jsou vněm uložena (číslo, text, logická hodnota, atd.).
  - Řádky tabulky reprezentují samotné záznamy (objekty) v databázové tabulce
- Každý řádek by měl mít svůj jedinečný identifikátor, podle kterého bude možné určit příslušný záznam – **klíč**.
  - Primární klíče slouží jako jednoznačný (unikátní) identifikátor záznamu (řádku) tabulky.
  - Cizí klíče slouží k vyjádření vztahů (relací) mezi jednotlivými tabulkami

SQL

- Standardizovaný jazyk relačních databází, vytvořený po představení relačního modelu
- Neprocedurální jazyk – je třeba zadat, jaké informace požadujeme, nikoli jak je získat
- Ve skutečnosti nejde o jeden jazyk, ale o různé dialekty různých databázových systémů (ORACLE, MS SQL, MySQL, ...)

SQL příkazy

- **DML** (Data Manipulation Language) – množina příkazů pro manipulaci s daty.
- **DDL** (Data Definition Language) – množina příkazů pro definici dat.
- **DCL** (Data Control Language) – množina příkazů pro definici řízení přístupových práv.
- **TCL** (Transaction Control) – množina příkazů pro řízení transakcí.

DML – příkazy pro manipulaci s daty

- SELECT – vybírá data z databáze.
- INSERT – vkládá data do databáze.
- UPDATE – edituje data v databázi.
- DELETE – odstraňuje data z celých tabulek nebo řádky tabulek odpovídající podmínce.
- CALL – volá uložené procedury.
- LOCK TABLE – zamyká celé tabulky:
  - zámky čtení (READ)
  - zámky zápisu (WRITE)

DDL – příkazy pro definici dabázových struktur

- CREATE – vytváří objekty databáze (i databázi samotnou).
- ALTER – mění strukturu databázových objektů.
- DROP – odstraňuje objekty databáze (tabulky, indexy, databáze).
- TRUNCATE – odstraní všechny záznamy z tabulky, přičemž zachová strukturu tabulky (objekt tabulky).
- COMMENT – umožní přidávat komentáře k objektům databáze.
- RENAME – změna názvu objektu databáze

DCL – příkazy pro řízení uživatelských práv

- GRANT – nastavuje uživatelská práva k tabulkám databáze.
- REVOKE – odebírá přístupová práva přidělená příkazem GRANT.

TCL – příkazy pro řízení transakcí

- COMMIT – potvrzení provedení transakce.
- SAVEPOINT – definuje záchytný bod transakce, ke kterému se lze během provádění transakce vrátit.
- ROLLBACK – ruší transakci a vrací se zpět ke stavu původnímu.
- START TRANSACTION – zahajuje blok transakce.
- SET TRANSACTION – umožňuje změnit level izolace transakce globálně nebo pro aktuální session.

Tabulky

Table: characters

id	name	home
1	John	Winterfell
2	Tyrion	Casterly Rock
3	Daenerys	Dragon Stone

Table: appearance

episode	character_id	appeared
1	1	10
1	1	5
1	1	7
2	1	5
2	2	5
2	3	9
3	1	10
3	2	2

Základní datové typy

- Integer (INT) – celé číslo
- Float (REAL) – raálná čísla
- Varchar (VARCHAR) – krátký řetězec do 255 znaků
- Text (TEXT) – delší text
- Binární data (BLOB) – obecná data, např. obrázky
- Boolean (BOOLEAN) – logická hodnota
- Date (DATE) – datum ve formátu YYYY-MM-DD
- Time (TIME) – čas ve formátu HH:mm:ss
- Timestamp (TIMESTAMP) – časová značka ve formátu YYYY-MM-DD HH:mm:ss

SQL – vytváření, změna a rušení tabulek

```
CREATE TABLE characters
(
id INT PRIMARY KEY,
name VARCHAR (255),
home VARCHAR (255)
);
```

```
ALTER TABLE characters MODIFY name VARCHAR (100) NOT NULL;
```

```
DROP TABLE IF EXISTS characters;
```

SQL – vkládání, mazání a změna řádků

```
INSERT INTO characters
VALUES (1, 'John', 'Winterfall');

INSERT INTO characters
VALUES (2, 'Tyrion', 'Casterly Rock');

INSERT INTO characters
VALUES (3, 'Daenerys', 'Dragon Stone');
```

id	name	home
1	John	Winterfall
2	Tyrion	Casterly Rock
3	Daenerys	Dragon Stone

```
DELETE FROM characters
WHERE name = 'John' OR id = 3;
```

id	name	home
2	Tyrion	Casterly Rock

```
UPDATE characters SET
home = 'Winterfell' WHERE id = 1
```

id	name	home
1	John	Winterfell

SQL – dotaz v jedné tabulce

```
SELECT id, name FROM characters
WHERE id < 3;
```

id	name
1	John
2	Tyrion

```
SELECT * FROM characters
WHERE id < 3;
```

id	name	home
1	John	Winterfell
2	Tyrion	Casterly Rock

SQL – dotaz ve více tabulkách, agregace

```
SELECT name, episode, appeared
FROM characters, appearance
WHERE id = character_id
AND episode = 1
AND appeared < 10
```

name	episode	appearance
john	1	5
daenerys	1	7

```
SELECT name, sum(appeared) as sum
FROM characters, appearance
WHERE id = character_id
GROUP BY id
```

name	sum
John	20
Tyrion	5
Daenerys	16

Transakce

Konzistence dat v databázi je zajištěnam pomocí transakcí. U relačních databází se jedná o princip **ACID** – Atomicity (atomicita), Consistency (konzistence), Isolation (nezávislost), Durability (trvanlivost).

Atomicita	Transakce musí proběhnout vcelku. Vyskytne-li se nějaká chyba v průběhu transakce, pak je celá transakce zrušena.
Konzistence	ransakce převede databázi z jednoho konzistentního stavu do jiného. Dojde k zápisu pouze platných dat, které dodržují integritní omezení
Nezávislost	Je-li spuštěno víc transakcí v jednom okamžiku, tyto transakce se navzájem neovlivňují. To znamená, že provedením částečné změny v databázi v průběhu transakce se projeví až po dokončení transakce.
Trvanlivost	dokončení úspěšné transakce jsou data uložena v databázi natrvalo.

Škálování

- Škálování je schopnost systému, sítě nebo procesu zvládnout narůstající množství operací nebo objemu dat.
- Při škálování databází existují dva přístupy:

vertikální

- Přidání dalších zdrojů v rámci stejné logické jednotky
- Další CPU, rozšíření RAM, HDD

horizontální

- Přidání dalších logických jednotek zdrojů – distribuované systémy
- Tradiční model vyvažování zátěže, základ cloudu
- Schopnost distribuovat jak data, tak zatížení jednoduchých operací na mnoha serverech bez sdílení RAM nebo diskového prostoru.
- Horizontální škálování je považováno za modernější přístup.

Index

- Umožňují rychlejší vyhledávání v tabulce a nalezení odpovídajícího objektu (řádku)
- Tabulka může obsahovat více klíčů
- Nevýhodou je zpomalení aktualizace a místo potřebné pro strukturu indexů.
- Existují čtyři základní typy indexů:

Index	Někdy označován jako sekundární. Je vytvářen ve sloupcích, kde se nenachází primární klíč. Slouží k efektivnímu hledání záznamů, které jsou jiné než primární klíč.
Primární	Tento index se vytváří ve sloupci, ve kterém se nachází primární klíč. V tomto sloupci je každá hodnota unikátní a žádná hodnota není neznámá.
Unikátní	Jedinečný index zajišťuje dostupnost pouze neopakujících se hodnot, a proto je každý řádek jedinečný.
Fulltextový	Podporuje efektivní vyhledávání slov v řetězcových datech.

Architektura databázových systémů

- Aplikační vrstva nebo vrstva uživatelského rozhraní představuje rozhraní pro všechny uživatelé
- Logická vrstva obsahuje základní funkce řídicího systému relační databáze (RDBMS). Tato část systému obsahuje celou řadu implementací specifických pro dodavatele. Vztahuje se ke konkrétnímu databázovému modelu a používá jeho konstrukční dotazovací a manipulační prostředek.
- Datové soubory, které uchovávají uživatelská data.
  - Indexy, které poskytují rychlý přístup k datovým položkám, které obsahují určité hodnoty.
  - Statistické údaje, které uchovávají statistické informace o datech v databázi. Je používán dotazovacím procesorem, k výběru efektivních způsobů provedení dotazu.
  - Logy, které se používají ke sledování provedených dotazů. Tyto data mohou být využita třeba při obnovení databáze v případě selhání systému.

Příklad relačních databáze – SQLite

- Jednoduchá databáze , která ukládá data v binární formě na lokální paměťové médium
- Podmnožina SQL jazyka
- Řádkový klient, podpora v řadě progamovacích jazyků včetně C, C++, Python, ...
- Uživatelský profil v prohlížeči Mozilla, lokální storage v Androidu, ...
- Celá řada GUI nástrojů: SQLiteStudio SQLite Database Browser

SQLite – vytvoření databáze

```
1  import sqlite3
2  from sqlite3 import Error
3
4  def create_connection(db_file):
5    """ create a database connection to a SQLite database """
6    conn = None
7    try:
8      conn = sqlite3.connect(db_file)
9      print("SQLite version: ", sqlite3.version)
10   except Error as e:
11     print("Error: ", e)
12   finally:
13     if conn:
14       conn.close()
15
16 if __name__ == '__main__':
17   create_connection("iot.db")
```

SQLite – vytvoření databáze

```
def db_exec(conn, sql):
  """ execute a SQL comman
  :param conn: Connection object
  :param sql:  SQL statement
  """
  try:
    c = conn.cursor()
    c.execute(sql)
  except Error as e:
    print(e)
sql_create_projects_table = """ CREATE TABLE IF NOT EXISTS values (
  id INTEGER PRIMARY KEY,
  timestamp DATETIME NOT NULL,
  temperature REAL NOT NULL,
  humidity REAL NOT NULL); """
```

SQLite – SQL dotazy

Vložení dat

```
insert into Si7021 (timestamp, temperature, humidity)
values (datetime(), 12.2, 65.5);
```

Získání dat

```
select * from Si7021
```

Formátování dat

```
select
  strftime('%Y-%m-%d %H:%M:%S', datetime()),
  temperature,
  humidity
from Si7021
```

SQLite – vložení dat v Pythonu

```
1  def db_insert(conn, data):
2      """
3      Create a new record
4      :param conn:
5      :param data:
6      :return:
7      """
9      sql = """ insert into Si7021 (timestamp, temperature, humidity)
           values (datetime(), ?, ?); """
10     cur = conn.cursor()
11     cur.execute(sql, data)
12     conn.commit()
14     return cur.lastrowid
```

SQLite – získání dat v Pythonu

```
1  def db_fetch (conn):
2    """
3    Fetch data from table
4    :param conn: the Connection object
5    :return:
6    """
7    cur = conn.cursor()
8    cur.execute("SELECT * FROM Si7021")
10   rows = cur.fetchall()
12   for row in rows:
13     print(row)
```

NoSQL databáze – vznik

- Vzrůstající objem dat
- Rostoucí propojitelnost dat
- Ztráta předvídatelné architektury
- Současná architektura aplikací
- ACID zbytečně restriktivní
- Neznáme dopředu přesnou strukturu dat
- Výpadek je normální a umíme na něj reagovat

Kdy nepoužívat NoSQL

- Potřebuji-li transakce
- Chci mít pevnou strukturu (schéma)
- Potřebuji spojovat tabulky

Rozdělení NoSql databází

- Dokumentové databáze (Document stored)
  - místo s řádkem pracují s dokumentem
  - CouchDB, MongoDB
- Databáze klíč – hodnota (Key-value)
  - důraz na rychlost
  - Memcached, Riak, Redis, Cassandra
- Sloupcové databáze (Column)
  - data se ukládají do sloupců, lze je přidávat, vhodné pro datové sklady
- Grafové (Graph database)
  - uzly a hrany – vztahy mezi sousedy – nalezení souvislostí

Key-value databáze

- Asociativní pole nebo hashovací tabulka s ukládáním hodnot podle unikátního klíče; jmenné prostory pro oddělení dat
- Serializovaná data (protokoly Apache Thrift, Protocol Buffers...)
- Rychlé vytvoření, rychlost práce s daty
- Základní API – operace GET, PUT, DELETE

Dokumentové databáze

- Datový model dokument – datová struktura se samopopisným charakterem
- Dokumenty používají pro ukládání dat i pro komunikaci s klienty
- Relační i objektové mapování
- Vhodné pro úlohy, kde je vazba na reálné měnící se dokumenty

Pojmy

- MapReduce – algoritmus k výpočtu pohledu, klient-server, dotazy
- B-tree – struktura
- JSON (JavaScript Object Notation) – zápis krátkých strukturovaných zpráv pro výměnu dat mezi aplikacemi (alternativa k XML)
- BSON – Binary JSON (http://bsonspec.org/)

JSON – JavaScript Object Notification

- Kolekce párů název/hodnota.
- 4 datové typy (řetězec, číslo, logická hodnota, null) a dva strukturované (objekt, pole)
- Rychlá práce s velkým objemem dat
- Jednoduchost; knihovny pro různé jazyky

```
{ "users":[
            {
                "firstName":"Ray",
                "lastName":"Villalobos",
                "joined": {
                    "month":"January",
                    "day":12,
                    "year":2012
                }
            },
            {
                "firstName":"John",
                "lastName":"Jones",
                "joined": {
                    "month":"April",
                    "day":28,
                    "year":2010
                }
            }
    ]}
```

BSON

JSON:

```
{
  "a": 3,
  "b": "xyz"
}
```

BSON: