Spatial databases, ORM

Přednáška č. 10

RDB

Historie SQL a relačního modelu dat

- Teoretický základ byl položen v 70 letech minulého století (cca 1970).
- První komerční produkt byl dostupný v roce 1979 Oracle Database in release 2.
- Jako nejzásadnější milníky ve vývoji lze označit:
  - Formalizaci ACID (Atomicity, Consistency, Isolation, Durability), které umožní vyhnout se případnému poškození dat
  - Standardizace SQL
- Na první pohled pak celá další desetiletí nebyl zaznamenán žádný významný pokrok ve vývoji SQL databází
- Díky tomu se SQL a relační model dat občas nazývá jako starý a nepoužitelný v současném světě.
- Nicméně díky tomuto času se relační model a SQL vyvinulo do velmi stabilních a výkonných produktů
- To že tento proces trval tak dlouho je celkem logické vyústění toho, že SQL je deklarativní jazyk (vývojář nespecifikuje, které kroky a jak se mají data získat ale popíše jaké chce získat data), tento přístup byl v 80 letech zcela mimo technické možnosti té doby
- SQL nemá jen schopnost získat data, ale má v sobě přímo zabudované konstrukty pro předzpracování a transformaci dat
- Díky tomu bylo a je SQL stále velmi hojně využíváno

NoSQL na scéně

- NoSQL je nový trend pro posledních několik let.
- Problém mezi tábory proSQL a proNoSQL spolu bojují a používají zejména názory typu:
  - SQL je starý, pomalý, drahá technologie
  - NoSQL jsou nehomogenní kolekce nových, vysoce rozšiřitelných volných software, kterým se obecně říká NoSQL
- Jednou možností proč došlo SQL k této pověsti je velké rozšíření ORM, které obecně degraduje SQL databázi na pouhé uložiště dat. SQL schopnost předzpracovat data je pak schována za novou vrstvou, kterou tyto mapovací nástroje poskytují. Díky tomu SQL přišlo o jednu ze svých hlavních výhod a tak se řada vývojářů obrátila k NoSQL.
- NoSQL ale nikdy nebylo míně jako proti SQL, ale spíše jako další možná alternativa k relačnímu modelu a jeho ACID pro speciální případy aplikací.
- CAP pak redukoval dostupnost klasického ACID teorému, kdy se ukázalo, že klasický ACID přístup nemůže využít virtuálně neomezených zdrojů dostupných v cloud prostředí.
- To je mimochodem to, co nabízí NoSQL místo velmi striktního ACID, které má za cíl držet data 100% konzistentní za každých okolností, NoSQL akceptují možné dočasné nekonzistence, díky čemuž se zvyšuje dostupnost v distribuovaném prostředí.
- Jednoduše řečeno NoSQL bude mít raději stará data než žádná data.
- Z toho ale vyplývá, že využití NoSQL je možné tam, kde nám tato nekonzistence nevadí. Typicky jsou to aplikace pro sociální média.
- Pokud je ovšem centrální uložení data dostatečné, pak tu není důvod opustit bezpečí ACID databáze

SQL?

- SQL je stále velmi dobrý a výkonný nástroj pro celou řadu problémů.
- Bez ACID je velmi těžké napsat spolehlivý software.
- Např. Google se snaží vyvinout databázi, která by dokázala spojit výhody SQL a ACID a distribuce (databáze F1 - Cloud Spanner)
- V článku o této db se mimo jiné píše:“we find developers spend a significant fraction of their time building extremely complex and error-prone mechanisms to cope with eventual consistency and handle data that may be out of date.”

NewSQL?

- NewSQL is a term coined by 451 Group analyst Matt Aslett to describe a new group of databases that share much of the functionality of traditional SQL relational databases, while offering some of the benefits of NoSQL technologies.
- NewSQL systems offer the best of both worlds:
  - the relational data model and ACID transactional consistency of traditional operational databases;
  - the familiarity and interactivity of SQL;
  - the scalability and speed of NoSQL.
- Some offer stronger consistency guarantees than are available with NoSQL solutions, although others limit this to ‘tunable’ consistency and thus aren’t fully ACID-compliant.
- Of course, there are differences among NewSQL solutions.
- SAP HANA handles modest transactional workloads, but without the benefit of native clustering.
- NuoDB is a cluster-first SQL database with a focus on cloud deployments, but throughput is poor.
- MemSQL is useful for clustered analytics but its tunable consistency falls down on ACID transactions.

Cloud Spanner

- Hybridní databáze kombinující nejlepší vlastnosti tradičních relačních databází scalable NoSQL systémy
- Měla by nabízet:
  - Distribuované SQL dotazy,
  - Transakčně konzistentní sekundární indexy,
  - Asynchronní změny ve schématu databáze,
  - Optimistické transakce,
  - Automatické nahrávání historie a je jí publikování
- Např. http://www.theregister.co.uk/2013/08/30/google_f1_deepdive/
- https://www.everys.com/articles/news/google-opens-its-globally-distributed-cloud-spanner-database-everyone-use
- https://www.scribd.com/document/165524809/Google-F1-database

Cloud Spanner

- Cloud Spanner is the first and only relational database service that is both strongly consistent and horizontally scalable.
- With Cloud Spanner you enjoy all the traditional benefits of a relational database:
  - ACID transactions,
  - relational schemas (and schema changes without downtime),
  - SQL queries,
  - high performance,
  - high availability.
- But unlike any other relational database service, Cloud Spanner scales horizontally, to hundreds or thousands of servers, so it can handle the highest of transactional workloads.
- With automatic scaling, synchronous data replication, and node redundancy, Cloud Spanner delivers up to 99.999% (five 9s) availability for your mission-critical applications.
- In fact, Google's internal Spanner service has been handling millions of queries per second from many Google services for years.

NoSQL?

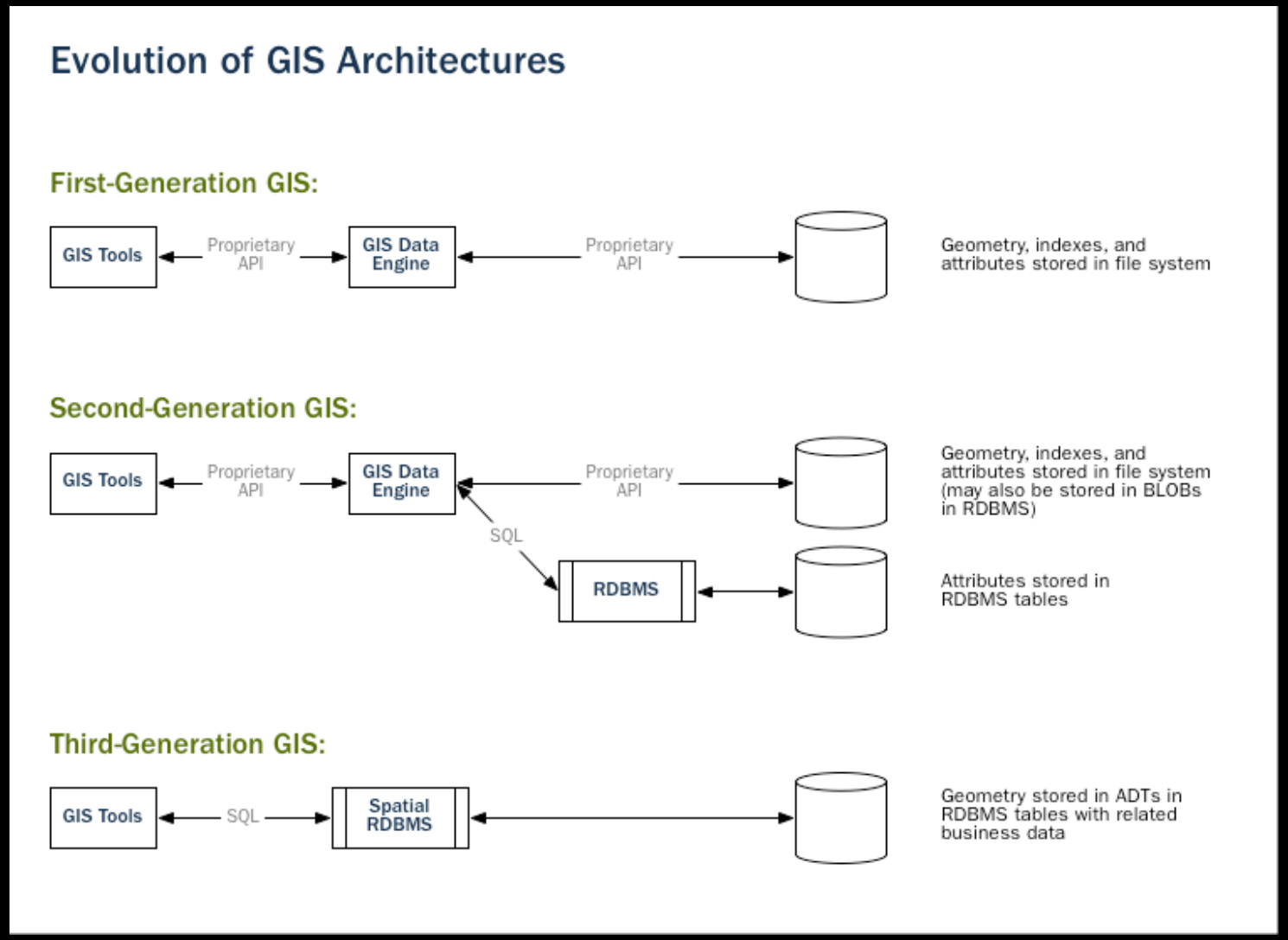
- Je tedy otázky, pokud se vrátíme k ACID a SQL co zbude z NoSQL?
- Jistotou je že SQL a ACID jsou stále jedním z hlavním kritérií a požadavků vývojářů databázových aplikací. Současně víme, že je možné udělat určité ústupky z ACID abychom získali scalability cloud prostředí (což by byl „svatý grál databází“).
- NoSQL systémy začaly tím, že přinesly neomezenou scalability a pak začaly přidávat nástroje pro kontrolu konzistence dat.
- Je tu např. i nový termín NewSQL, který se občas používá pro nové SQL databáze, které mají ACID ale interně využívají NoSQL myšlenky zlepšení scalability.
- Co nato „velcí“ výrobci databázových technologií?
  - Snaží se nás přesvědčit že mají i NoSQL řešení jako jsou například “Oracle NoSQL Database” nebo “Windows Azure Table Storage Service.”
  - Při pohledu např. na poslední verzi Oracle 12*c*, můžeme vidět docela opačný trend než je NoSQL. C v názvu by mělo znamenat cloud capabilities.
  - Nicméně hlavním přínosem této verze se jeví být: snadné a bezpečné provozování mnoha databází na jednom serveru.
  - Virtualizace se jeví být daleko žhavější trend než tzv. scale-out.
- Jak je to možné?
- Je tedy otázka jestli NoSQL není vhodné jen pro malou část trhu (jakou jsou např. firmy typu of Google, Facebook, Twitter atd.)
- Jednou z možností je fakt, že NoSQL řeší problém velkých dat, který je velmi populární a „každý by ho chtěl mít“.

Spatial databases - PostGIS

- Databáze pro uložení geografických dat (OpenMaps)
- Spatial databases store and manipulate spatial objects like any other object in the database.
  1. Spatial data types refer to shapes such as point, line, and polygon;
  2. Multi-dimensional spatial indexing is used for efficient processing of spatial operations;
  3. Spatial functions, posed in SQL, are for querying of spatial properties and relationships.

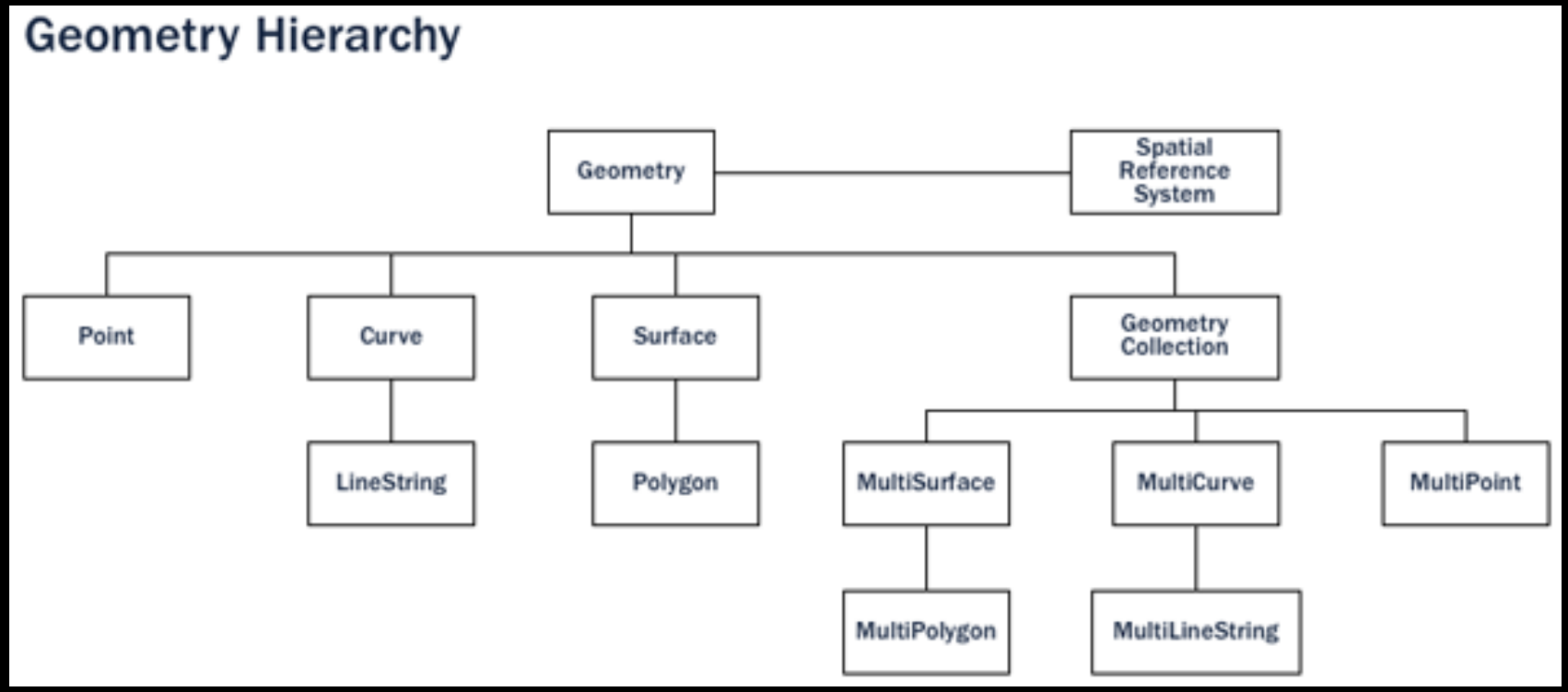
Zdroj: http://workshops.boundlessgeo.com/postgis-intro/introduction.html

GIS architektury



1. První generace- vše uloženo s souborech a specifický GIS software pak se soubory pracoval.
2. Druhá generace – část dat uloženo v relačních databázi, stále ale není dostupná flexibility v manipulaci s daty
3. **Třetí generace – spatial data jsou uložena jako objekty v databázi.**

Spatial Data Types



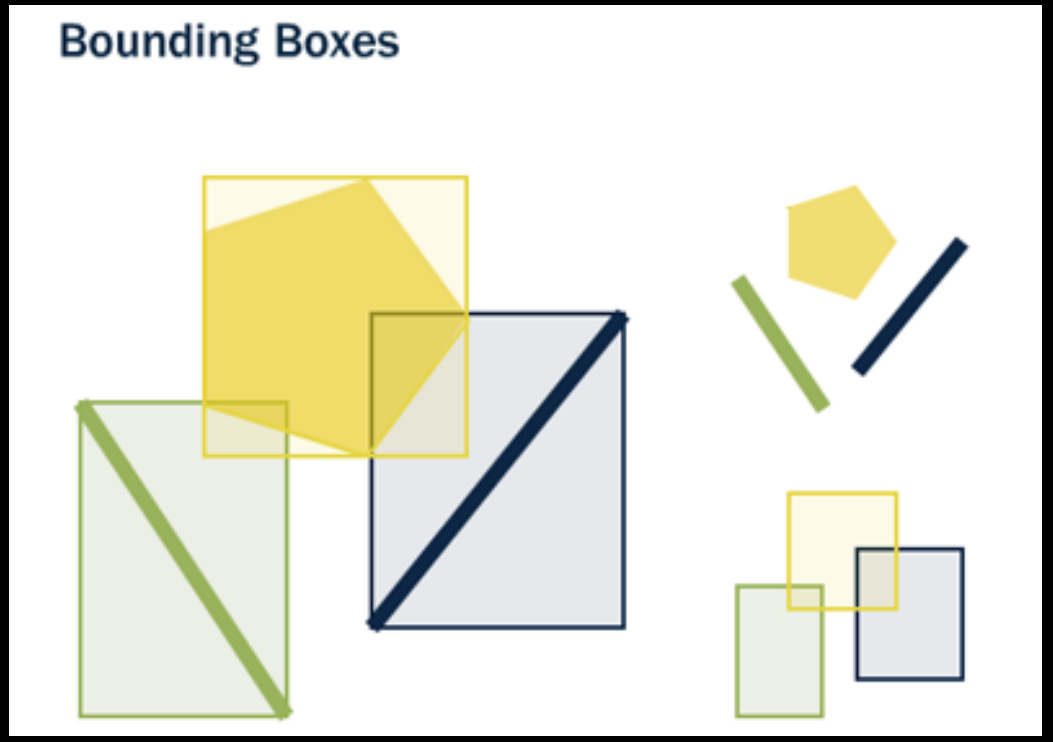
Zdroj: http://workshops.boundlessgeo.com/postgis-intro/introduction.html

Spatial Indexes and Bounding Boxes

- Klasické databáze umožňují optimalizovat přístup k datům pomocí indexů
- Problém je, že v geografických datech se můžou určité regiony překrývat atd. (nefunguje jednoduché porovnání menší, vetší, rovno), což je pro klasické indexování problém.
- Proto se používají tzv. “spatial index”, které slouží k vyhledávání typu: “které objekty jsou uvnitř určitého nadřazeného objektu (bounding box)?”.
- **bounding box** je nejmenší obdélník, který má danou vlastnost

Spatial Indexes and Bounding Boxes

- Bounding boxes pak odpovídají na otázku je A uvnitř B?, což je celkem výpočetně náročně pro polygony ale snadné pro obdélníky.
- Indexy musí být rychlé, takže místo přesné odpovědi kterou dávají B stromy, spatial indexy dávají přibližnou odpověď.
- Otázka “které linie jsou uvnitř daného polygonu?” bude reprezentována pomocí spatial indexu jako “které linie mají bounding boxes, které jsou uvnitř bounding box dotazovaného polygonu?”
- Nejčastěji se používají R-stromy, nicméně jsou využívány i Quadtrees a grid-based indexes



Zdroj: http://workshops.boundlessgeo.com/postgis-intro/introduction.html

Spatial Functions

- Databázové funkce pro analýzu geometrický dat a jejich vztahů
- Základní skupiny funkcí:
  - **Konverze**: konverze mezi geometrickými data a externími formáty.
  - **Management**
  - **Získávání**: získávání vlastností geometrie.
  - **Porovnání**: porovnání tvarů vzhledem k jejich prostorovým vztahům.
  - **Generování**: generování nových tvarů z jiných.

Ukázky funkcí

- Multi-geometry
  - ST_NumGeometries(geometry) Returns the number of geometries included in a multi-geometry or geometry collection.
  - ST_GeometryN(geometry, index) Returns the geometry at the given index within the multi-geometry or geometry collection, or null if index is greater than the number of sub-geometries. The first geometry is at index 1, the last at the value of index.
- Polygon
  - ST_NumInteriorRings(geometry) Returns the number of interior rings contained in the given polygon.
  - ST_NRings(geometry) Returns the number of rings, including the exterior ring, in the given polygon.
  - ST_ExteriorRing(geometry) Returns the exterior ring of the geometry as a Linestring.
  - ST_InteriorRingN(geometry, index) Returns the ring at the given index within the polygon as a Linestring or null if index is greater than the number of rings. The first interior ring is at index 1, the last at the value of index.
- LineString
  - ST_NumPoints(geometry) Returns the number of points contained in a Linestring geometry.
  - ST_NPoints(geometry) Like ST_NumPoints but usable against any geometry type.
  - ST_PointN(geometry, index) Returns the point at the given index within the linestring or null if index is greater than the number of points. The first point is at index 1, the last at the value of index.
- Metody pro získání začátku a konce bodů úsečky.
  - ST_StartPoint(geometry) = ST_PointN(geometry, 0)
  - ST_EndPoint(geometry) = ST_PointN(geometry, ST_NumPoints(geometry))
- Point
  - ST_X(geometry) ST_Y(geometry) ST_Z(geometry) ST_M(geometry) These functions return the x, y, z, and m coordinate of the given point respectively. In the case of ST_Z and ST_M, null is returned when the coordinate is not available.

Ukázky dotazů

- Let’s take our Broad Street subway station and determine its neighborhood using the **ST_Intersects** function:

```
SELECT name, ST_AsText(geom)FROM nyc_subway_stations
WHERE name = 'Broad St';
```

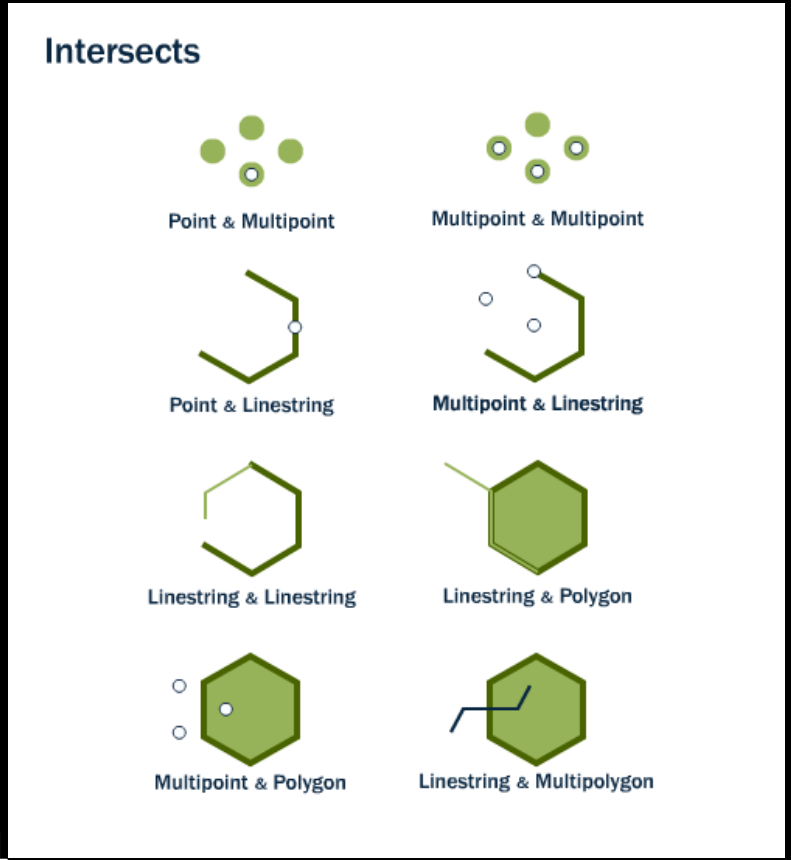
POINT(583571 4506714)

```
SELECT name, boroname FROM nyc_neighborhoods
WHERE ST_Intersects(geom,ST_GeomFromText('POINT(583571
4506714)',26918));
```

---

name | boroname

Financial District | Manhattan:



Ukázky dotazů II

- ST_Distance(geometry A, geometry B) calculates the shortest distance between two geometries and returns it as a float.

```
SELECT ST_Distance(
ST_GeometryFromText('POINT(0 5)'),
ST_GeometryFromText('LINESTRING(-2 2, 2 2)'));


--------------
3
```

Ukázky dotazů III

- find the streets nearby (within 10 meters of) the subway stop

```
SELECT name FROM nyc_streets WHERE ST_DWithin(geom,
ST_GeomFromText('POINT(583571 4506714)',26918),10);
```

---

name

Wall St

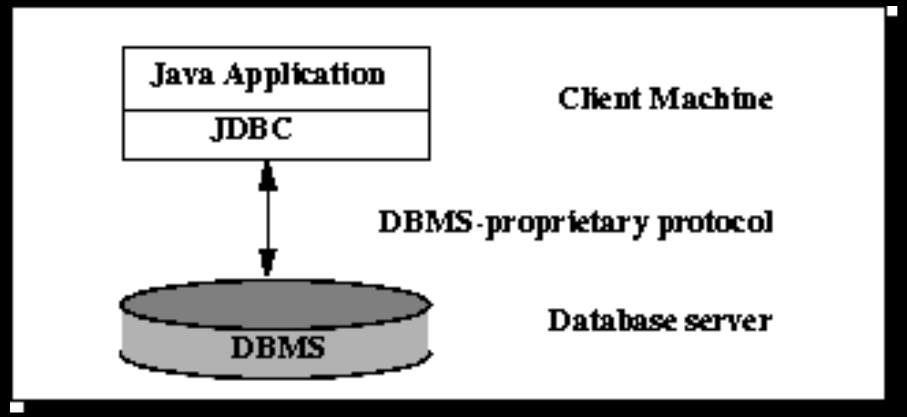
Broad St

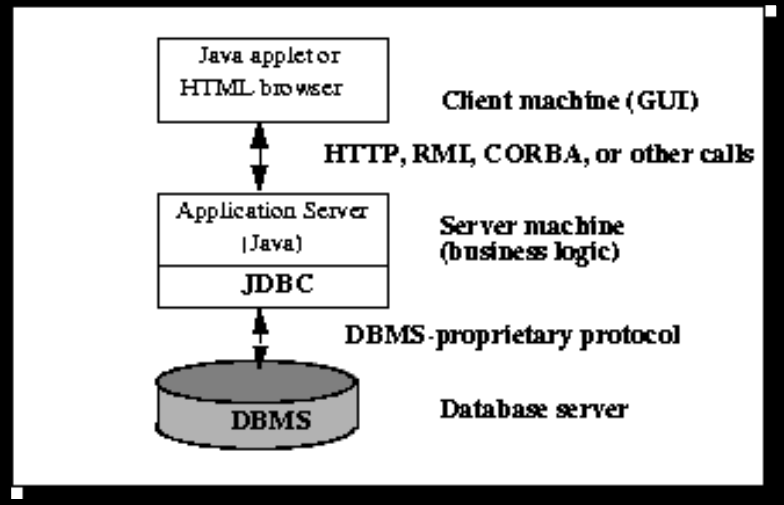
Nassau St

JDBC – Java Database Connectivity

- JAVA SE knihovna pro dotazování a aktualizaci databázových objektů/dat
- Zaměřena na relační model dat
- Spravuje připojení k DB buď přímo nebo pomocí knihoven třetích stran
- Tvůrci Db pak poskytují vlastní ovladače pro JDBC

JDBC – architektura





JDBC

- API poskytuje abstrakci nad interakcemi, datovými typy atd.
- Spuštění SQL pomocí „Statements“
- Výsledky dotazů pak pomocí „ResultSet“
- Transakční
- Cache

ORM – Object/Relational Mapping

- Persistence Datové objekty přetrvají i po běhu Java aplikace
- Mapuje Java POJO objety na relační databázi

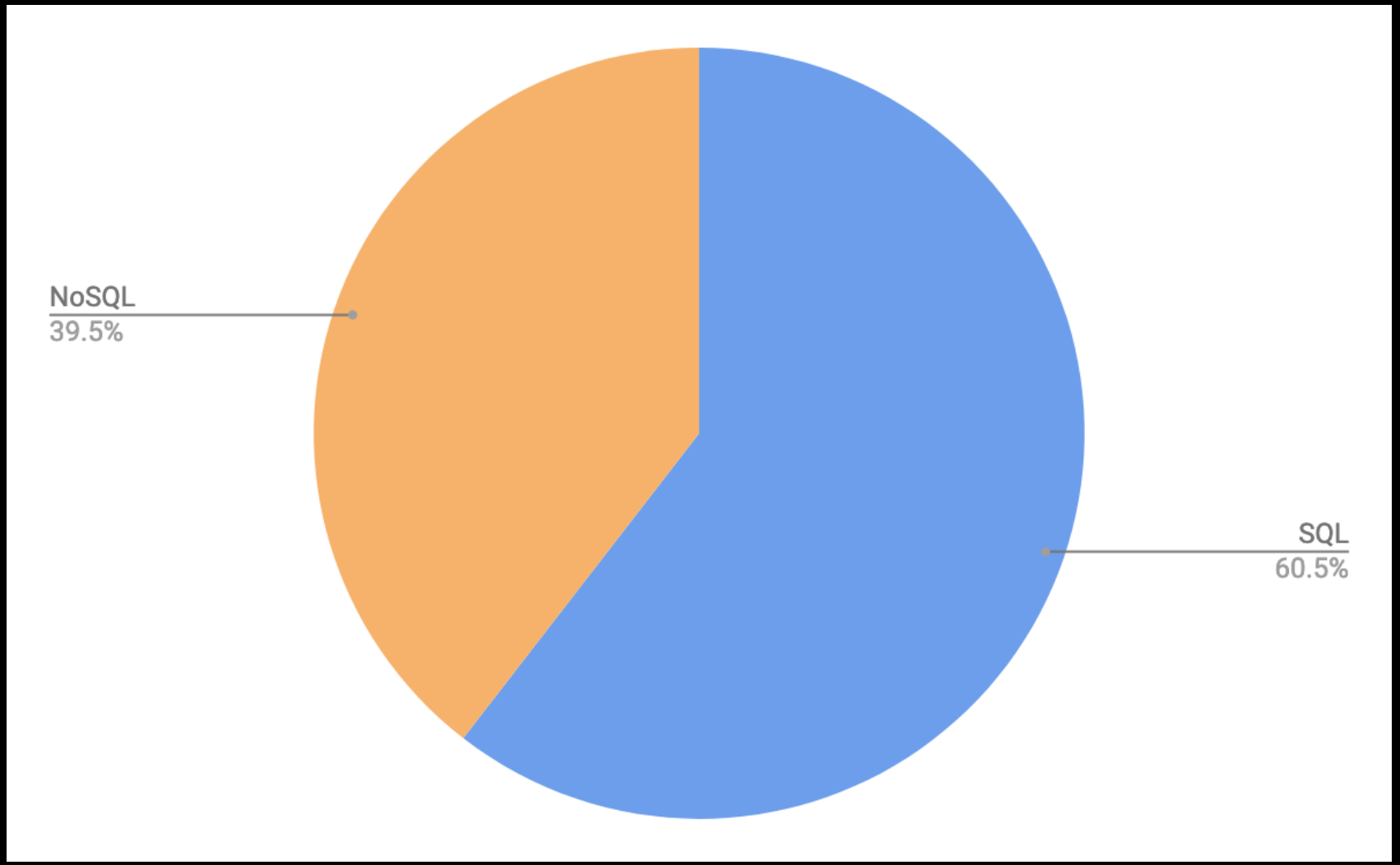
Proč ORM?

- Vzor „Domain Model“:
  - Zaměřuje se na koncepty funkce aplikace (Bussiness concepts) ne na strukturu relační databáze
  - Propojené objekty
  - Každý objekt je smysluplný nebo je konceptem
  - OO koncepty: dědičnost, identifikace objektu, atd.
  - Navigace v datech pomocí procházení grafu objektů – ne přímo relačního modelu dat

Proč ORM?

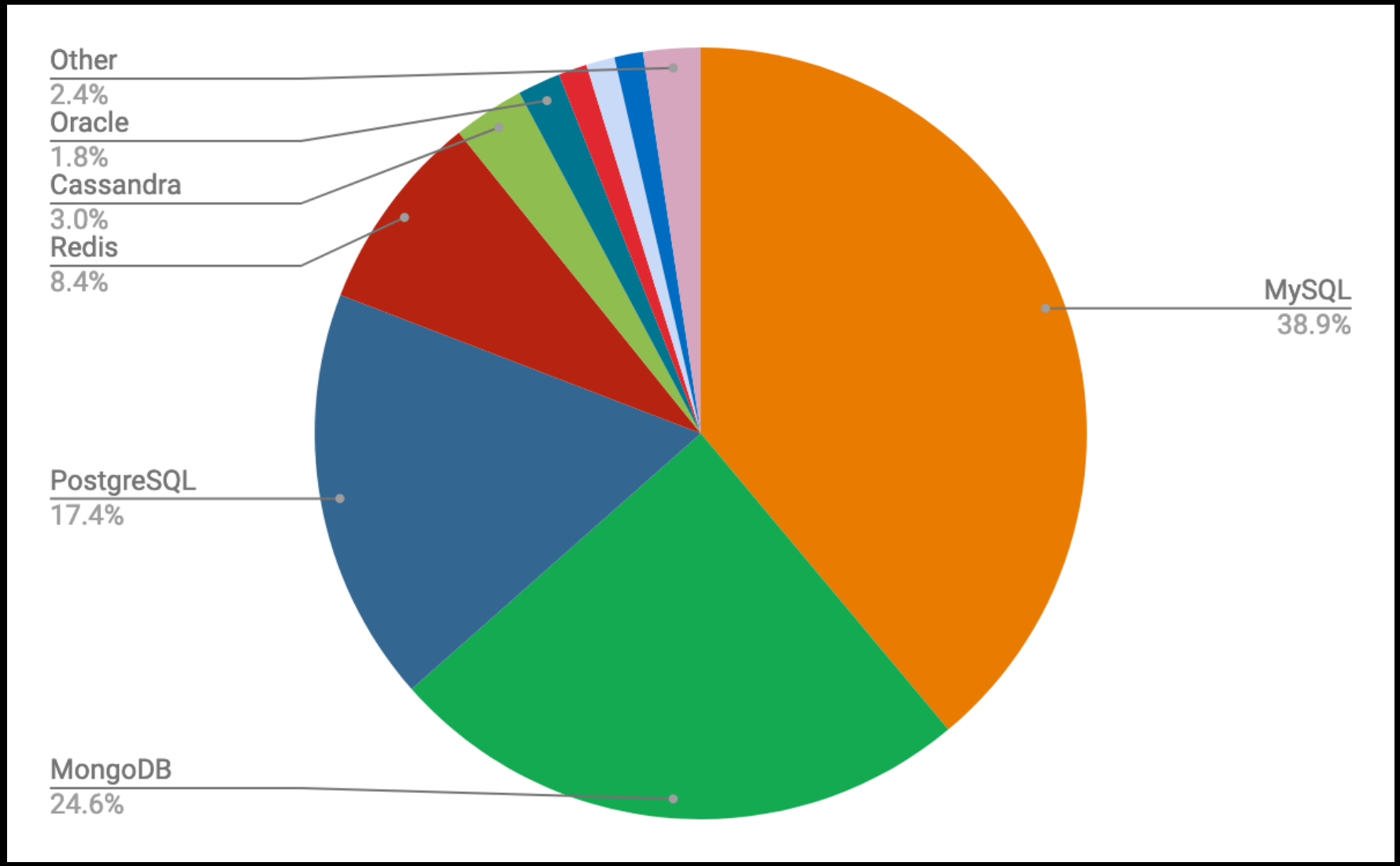
- Zrychluje vývoje a redukuje kód
  - Žádné ruční mapování JDBC ResultSet na POJO
  - Méně práce se synchronizací kódu s relační databází (změnami ve struktuře)
  - Méně opakování základních JDBC kódu pro zpracování dotazů
  - Zaměřen na logiku programu
- Portability
  - Nezávislé na DB (až na pár výjimek jako např. typy vlastností, generování identifikátorů)
  - Abstrakce dotazování (OO API, OO strukturované jazyky)
  - SQL specifické pro konkrétní DB je generováno

Trendy v DB 2019 – SQL vs NoSQL



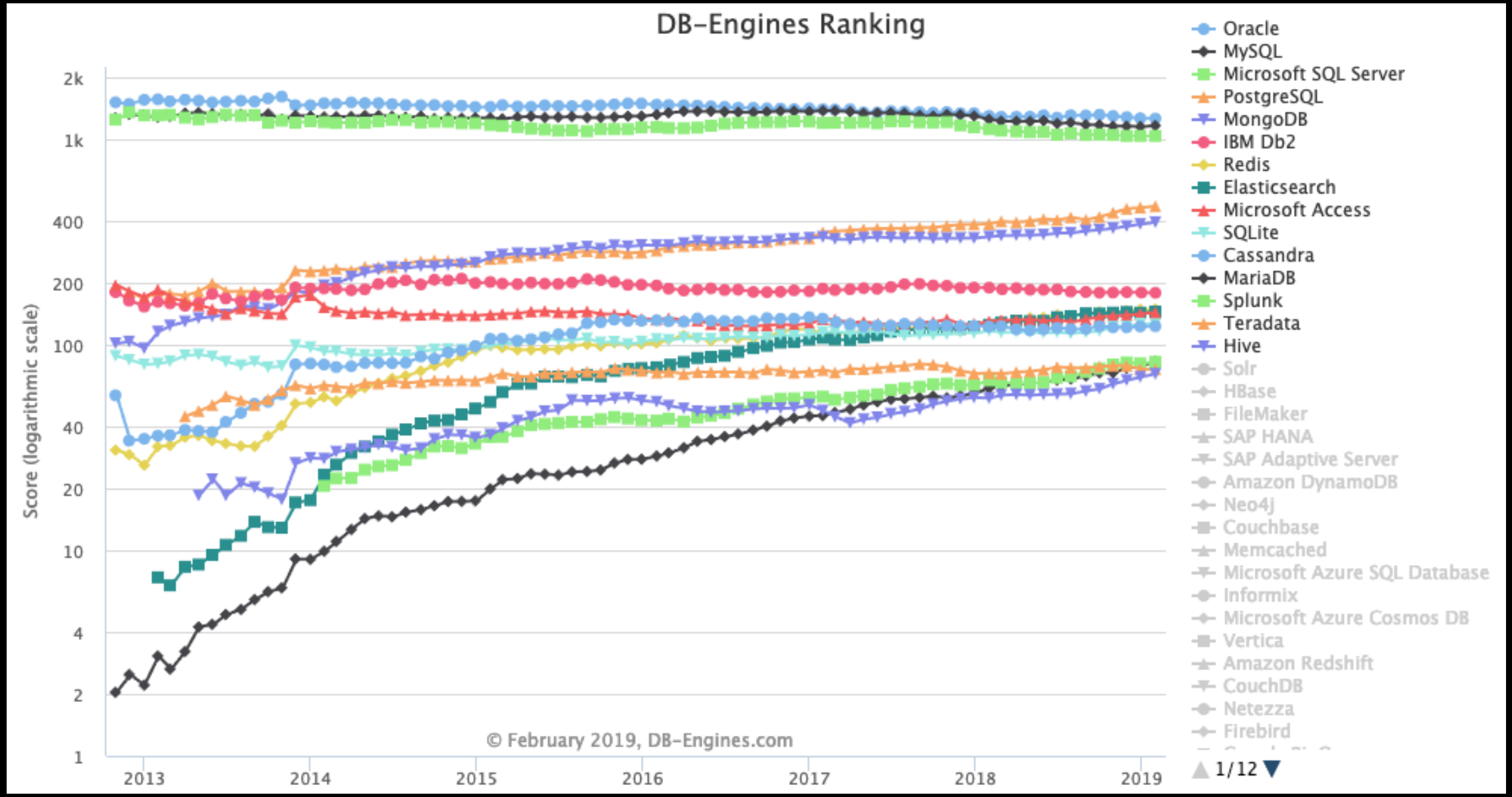
http://highscalability.com/blog/2019/3/6/2019-database-trends-sql-vs-nosql-top-databases-single-vs-mu.html

Trendy v DB 2019 – Nejpopulárnější DB



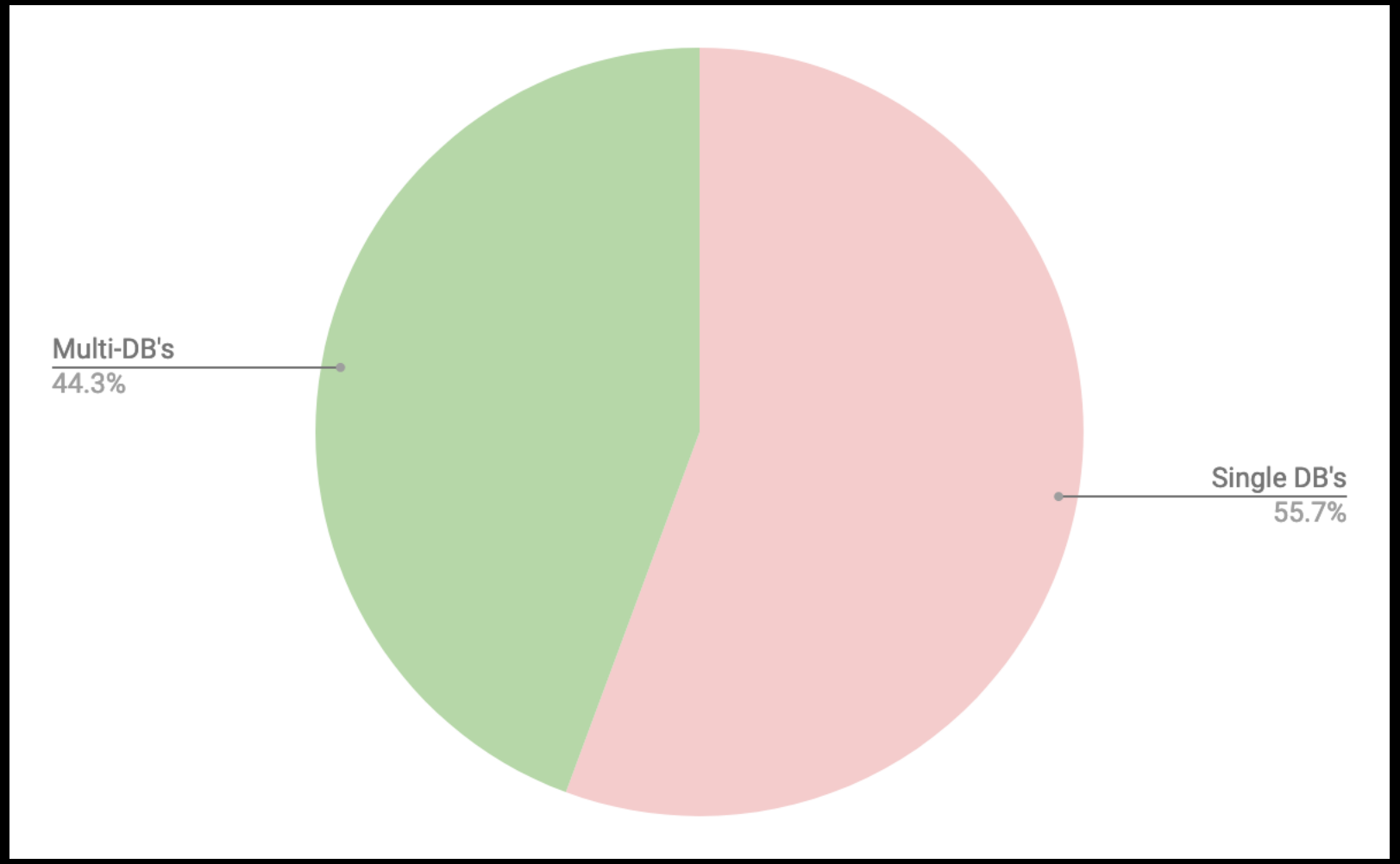
http://highscalability.com/blog/2019/3/6/2019-database-trends-sql-vs-nosql-top-databases-single-vs-mu.html

Trendy v DB 2019 – Nejpopulárnější DB



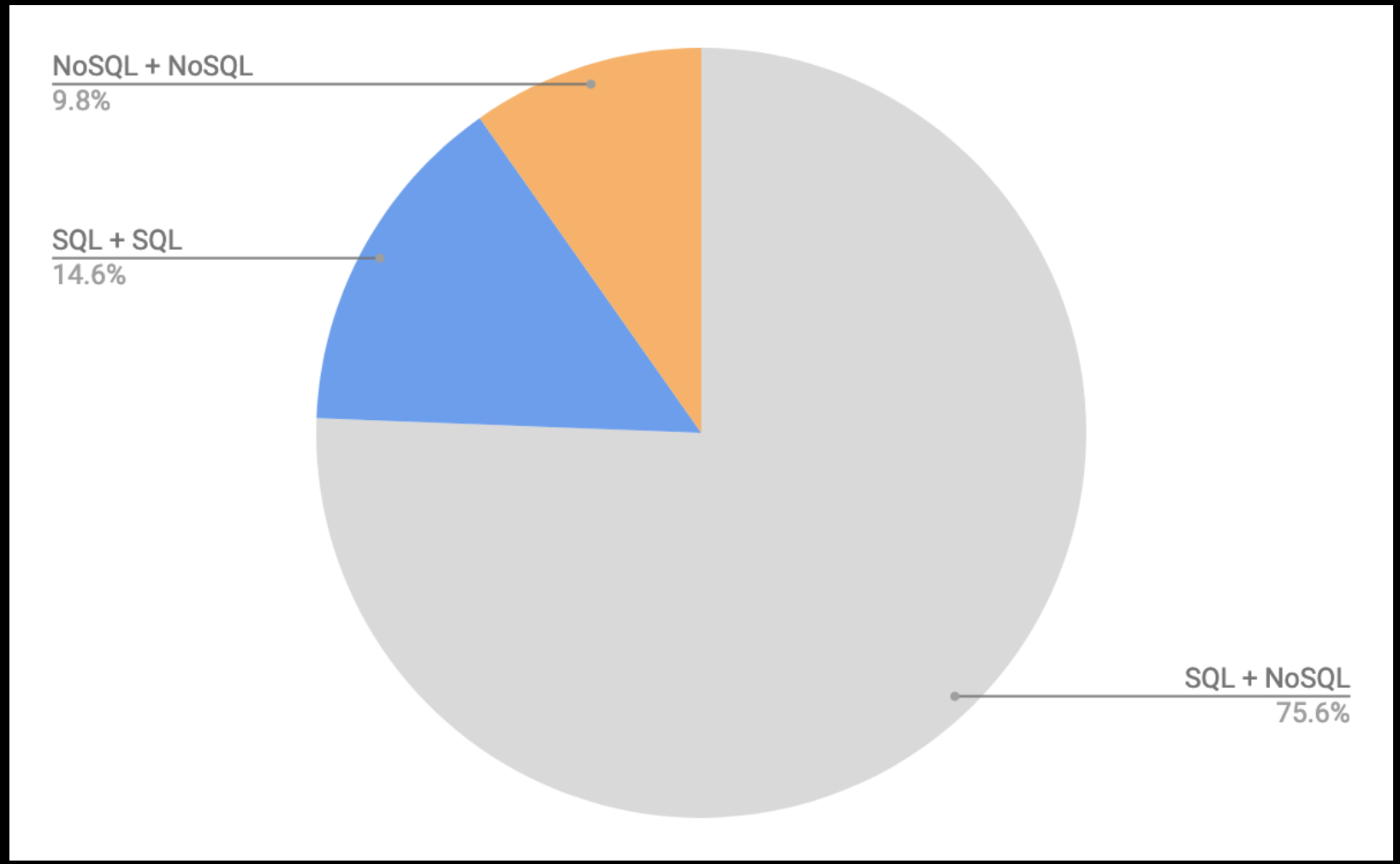
http://highscalability.com/blog/2019/3/6/2019-database-trends-sql-vs-nosql-top-databases-single-vs-mu.html

Trendy v DB 2019 – Single vs Multiple db server



http://highscalability.com/blog/2019/3/6/2019-database-trends-sql-vs-nosql-top-databases-single-vs-mu.html

Trendy v DB 2019 – SQL & NoSQL kombinace



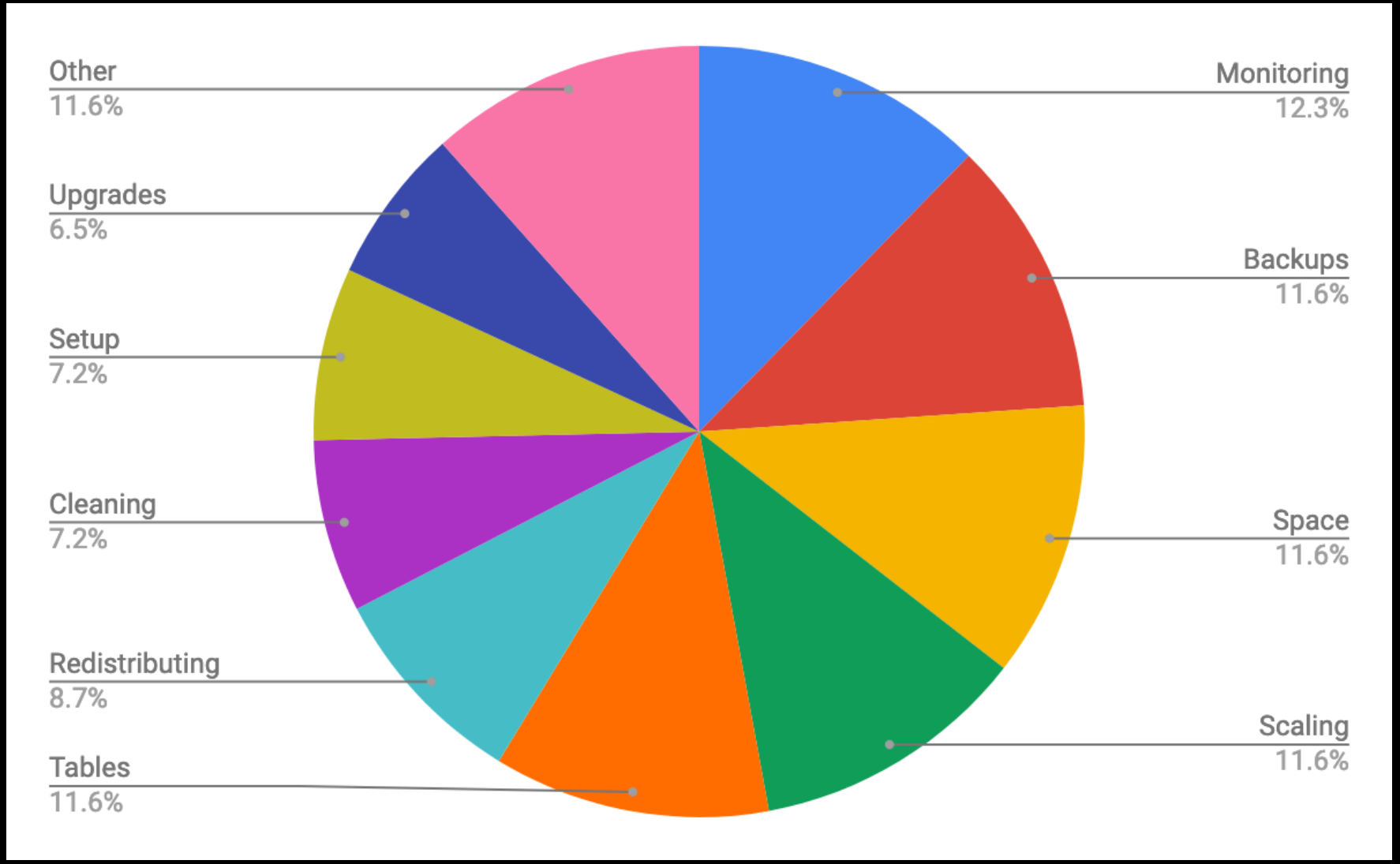
http://highscalability.com/blog/2019/3/6/2019-database-trends-sql-vs-nosql-top-databases-single-vs-mu.html

Trendy v DB 2019 – Nejpopulárnější kombinace

- MySQL + MongoDB: 34.15%
- MySQL + PostgreSQL: 9.76%
- MongoDB + PostgreSQL: 7.32%
- MongoDB + Redis: 7.32%
- MySQL + MongoDB + PostgreSQL: 4.88%
- MySQL + MongoDB + PostgreSQL + Redis: 4.88%

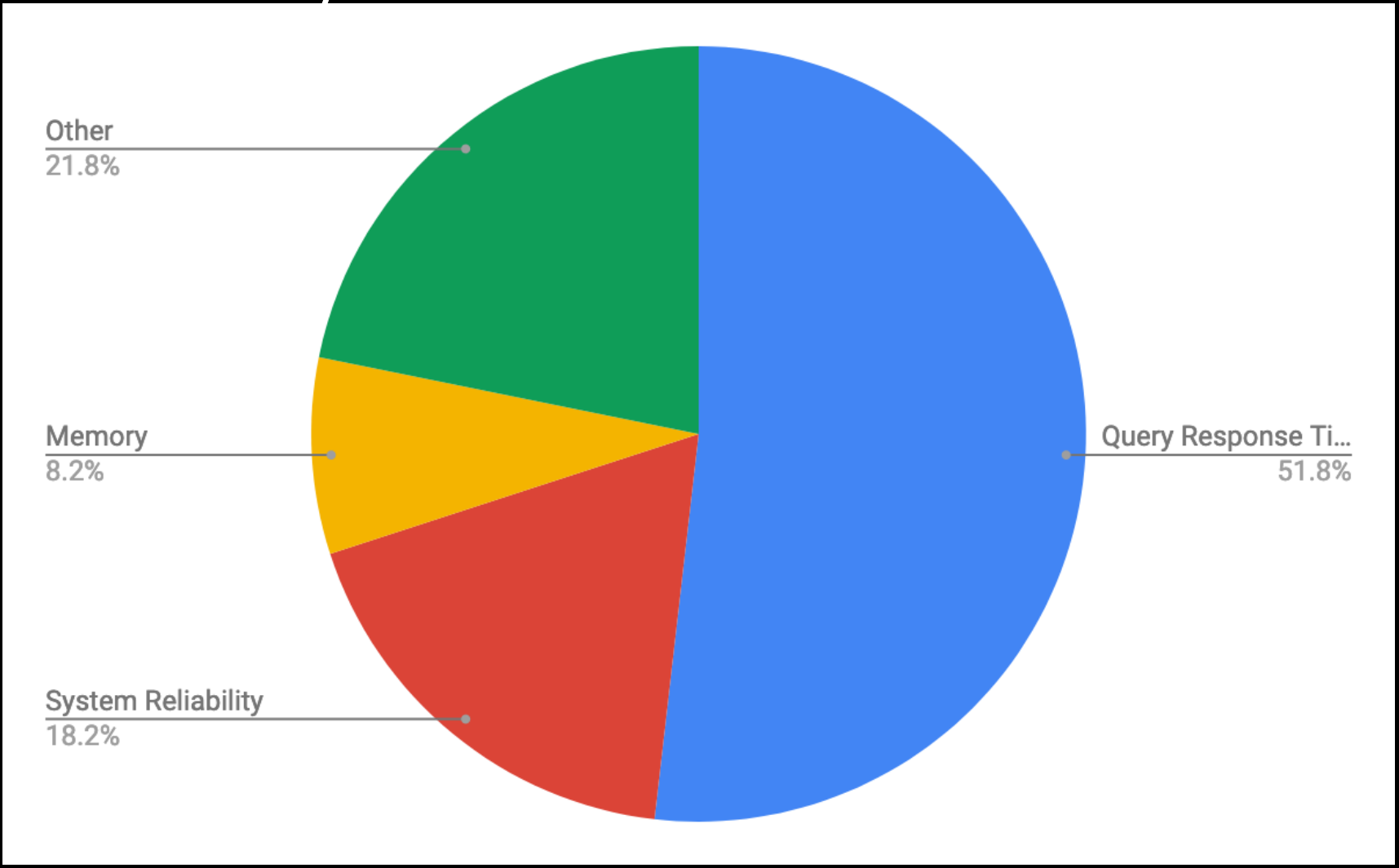
http://highscalability.com/blog/2019/3/6/2019-database-trends-sql-vs-nosql-top-databases-single-vs-mu.html

Trendy v DB 2019 – Nejdéle trvající úlohy



http://highscalability.com/blog/2019/3/6/2019-database-trends-sql-vs-nosql-top-databases-single-vs-mu.html

Trendy v DB 2019 – Nejvíce sledovaná metrika pro měření výkonu DB



http://highscalability.com/blog/2019/3/6/2019-database-trends-sql-vs-nosql-top-databases-single-vs-mu.html

Proč ORM?

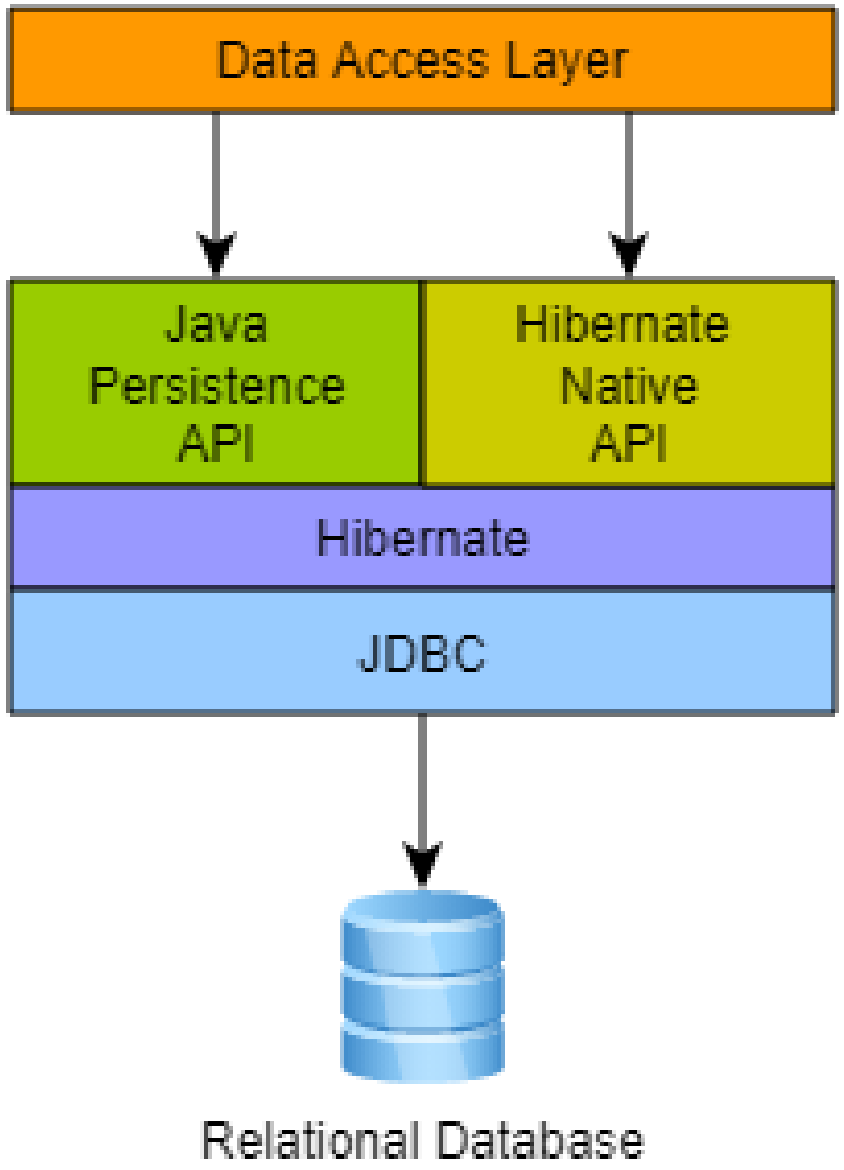
- Výkon:
  - Granulární kontrola kdy, jak a kolik dat/vztahů se načte v závislosti na logice programu
  - Cache objektů a dotazů
- Concurrency & multiple dependency
- Transakce
- Scalable
- Rozšiřitelnost

JPA – Java Persistence API

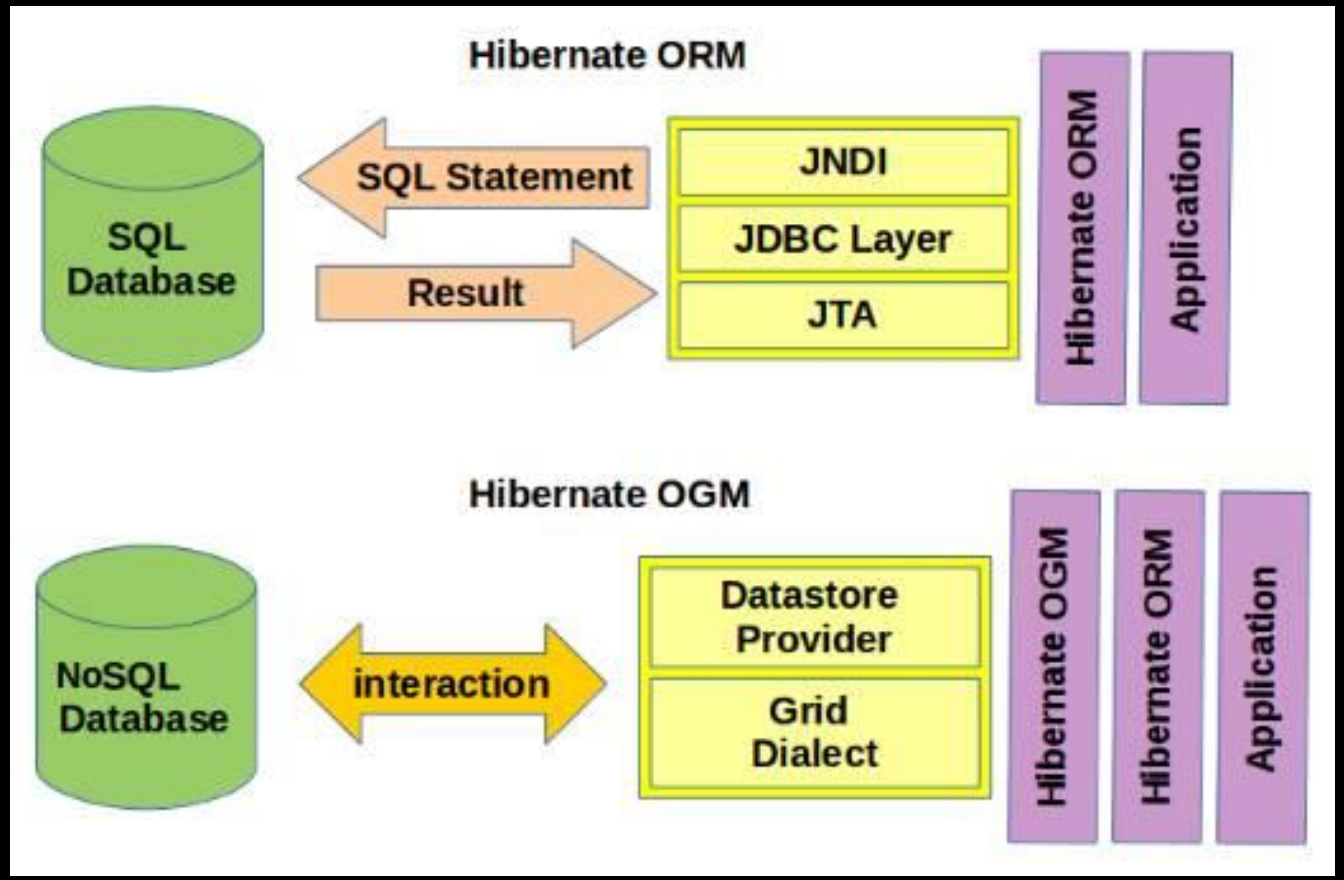
- Obsaženo v Java SE i Java EE
- Poskytuje propojitelnost mezi JPA implementacemi

Hibernate ORM

- JPA implementace
- vlastnosti:
  - HQL (Hibernate Query Language) – podobné jako JPQL ale má vlastnosti navíc
  - API pro kritéria
  - Výkon: strategie pro získání objektů (fetch, cachinig, bytecode, rozšíření, …)
  - Body pro další rozšíření
  - Nástroje pro validaci schémat a jejich generování



- http://docs.jboss.org/hibernate/orm/5.2/userguide/html_single/Hibernate_User_Guide.html



https://www.developer.com/java/data/how-to-manage-data-persistence-with-mongodb-and-jpa.html

Hibernate

- Hibernate Spatial
- Hibernate Envers
- Hibernate OSGi
- Hibernate Search
- **Hibernate OGM - NoSQL**
- Hibernate Validator
- Hibernate Shards

Příklad - MySQL

```
CREATE TABLE `employee` (
`id` int(11) NOT NULL AUTO_INCREMENT,
`emp_name` varchar(100) DEFAULT NULL,
`emp_address` varchar(500) DEFAULT NULL,
`emp_mobile_nos` varchar(100) DEFAULT NULL,
PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Maven - Správa závislostí

```
<project xmlns="http://maven.apache.org/POM/4.0.0"   xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance„ …>
…
<dependencies>
            <dependency>
              <groupId>mysql</groupId>
              <artifactId>mysql-connector-java</artifactId>
              <version>5.1.6</version>
            </dependency>

             <dependency>
              <groupId>org.hibernate</groupId>
              <artifactId>hibernate-core</artifactId>
              <version>5.2.8.Final</version>
             </dependency>
 </dependencies>
```

Maven

```
 <build>
<plugins>
     <plugin>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.1</version>
      <configuration>
       <source>1.8</source>
       <target>1.8</target>
      </configuration>
     </plugin>
   </plugins>
  </build>
```

```
@Entity
@Table(name = "employee")
public class Employee implements Serializable{
        @Id
        @GeneratedValue(strategy = GenerationType.IDENTITY)
        @Column(name="id")
        private int id;

        @Column(name="emp_name")
        private String empName;
        @Column(name="emp_address")
        private String empAddress;

        ....
}
```

Anotace nebo XML konfigurace

```
@Entity
@Table(name = “author”, schema = “bookstore”)
public class Author {

	@Id
	@GeneratedValue(strategy = GenerationType.SEQUENCE,
generator = “author_generator”)
	@SequenceGenerator(name=”author_generator”,
sequenceName = “author_seq”)
	@Column(name = “author_id”)
	private Long id;

	…
}
```

```
<entity-mappings>
   <entity class="org.thoughts.on.java.model.Author"
name="Author">
       <table name="author" />
       <schema name="bookstore" />
      <attributes>
         <id name="id">
             <generated-value strategy="sequence"
generator="author_generator"/>
             <column name="author_id"/>
         </id>
      </attributes>
  </entity>
  <sequence-generator name="author_generator"
sequence-name="author_seq"/>
</entity-mappings>
```

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-5.0.dtd">
<hibernate-configuration>
    <session-factory>
        <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
        <property name="hibernate.connection.url">jdbc:mysql://localhost:3306/hibernate5</property>
        <property name="hibernate.connection.username">root</property>
        <property name="hibernate.connection.password">root</property>
        <property name="hibernate.connection.pool_size">10</property>
        <property name="dialect">org.hibernate.dialect.MySQLDialect</property>
        <property name="hibernate.current_session_context_class">thread</property>
        <mapping class="model.Employee" />
    </session-factory>
</hibernate-configuration>
```