1. Co je to cost-based optimalizace a jak se využijí statistiky o databázových objektech při cost-based optimalizaci?

Pro každý dotaz jsou vytvořeny prováděcí plány *(posloupnost čtení dat, řazení, joinů, projekcí a selekcí)*, které mají různou cenu *(počet I/O bloků, které je nutno přečíst nebo zapsat)*. DB engine vytvoří různé plány, u každého spočítá jeho cenu na základě statistik o DB objektech, a vybere ten s nejnižší cenou.

2. Jak vypadá zpracování SQL dotazu (fáze zpracování dotazu, kde a jak se při nich dá optimalizovat)?

Dotaz se nejprve musí **naparsovat** (provede se syntaktická analýza, vytvoří se prováděcí plány, z nich se vybere nejlepší [*optimalizace*], zkontrolují se přístupová práva), následně se dotaz provede (**execution**) – v této fázi lze optimalizovat přístup na disk a jsou vráceny výsledky (**fetch**). Pokud byl proveden např. INSERT/UPDATE, s odstupem jsou pak aktualizovány statistiky o DB objektech.

3. Vysvětlete rozdíl mezi heap table a index-organized table.

Heap tabulka je „pole řádků“, každý řádek je identifikován svým ROWID, který identifikuje fyzické uložení dat. Řádky se po vložení nepřesouvají. V případě použití indexu na sloupci *col* se ukládá ROWID a data sloupce *col*.
Indexem organizovaná tabulka je uspořádána podle indexu. Při vkládání řádku může dojít ke změně fyzického umístění, data jsou uložena přímo v listech B-stromu.

4. Vysvětlete rozdíl mezi heap table a cluster.

U **heap tabulky** data nejsou seřazená, při přístupu přes index se tedy může stát, že budeme číst více I/O bloků.
Cluster obsahuje podobná data uložená spolu, pokud tedy provádíme čtení dat podle indexu, bude dotaz rychlejší.

5. Vysvětlete rozdíl mezi B-tree a bitmap indexem, příklady vhodného použití obou typů indexů.

B-tree index je vhodný pro data s vysokou kardinalitou *(hodně různých hodnot)*, jako je jméno, telefonní číslo. Strukturou je B-strom.
Bitmap index je vhodný pro data s nízkou kardinalitou *(málo různých hodnot)*, například pohlaví, kategorie. Strukturou je 2D pole s 0/1 hodnotami.

6. Jaké jsou typické statistiky pro tabulky v relační databázi a jak se udržují, když se pomocí DML mění data?

Počet řádků **nR**, blokovací faktor **bR** *(kolik řádků se vejde do jednoho bloku/stránky)*,
počet bloků/stránek **pR**, variabilita **V(A, R)** *(počet různých hodnot hodnoty A v relaci R)*.
Dále **min**/**max** hodnoty, či **histogram** rozmístění hodnot *(využití: dotaz typu WHERE x > 23)*.
Aktualizace probíhá pomocí démona, když je DB méně vytížená *(nikdy neprobíhá online)*.

7. Jaké jsou typické statistiky pro B-tree indexy a jak se udržují, když se pomocí DML mění data?

Faktor větvení **f(A,R)** *(kolik synů má uzel stromu, 50-150)*, hloubka stromu **I(A,R)** *(typicky 2-3)*, počet listových bloků **p(A, R)**.
Dále clustering faktor, složený index, reverzní index nebo index založený na funkci.
Aktualizace probíhá pomocí démona, když je DB méně vytížená *(nikdy neprobíhá online).*

8. Co jsou to přístupové cesty (access paths) při vyhodnocování SQL dotazů? Uveďte příklady.

Access path: Způsob, kterým DB systém získá požadované řádky z relace.
FULL TABLE SCAN / SEQUENTIAL SCAN – přečte se celá tabulka, nevyužívá se index.
INDEX SCAN – využití indexu, může vést k zrychlení dotazu

9. Jaké znáte metody vyhodnocení spojení (join) v relačních databázích? Naznačte jak probíhají.

Hnízděné cykly (nested loop): pro každý záznam z R iteruji každý záznam z S, hledám shodu
Merge join: seřadím relaci R, relaci S, a následně hledám shodu, lze využít prioritní frontu
Hash join: aplikuji hashovací funkci na atribut, vytvořím skupiny podle hashe, porovnávám.

10. Co to je prováděcí plán (execution plan), jak vypadá a kdy vzniká? Vyplatí se ho cachovat? Pokud ano, za jakých okolností?

Plán: strom operací, které DB engine vykonává: čtení dat, řazení, joiny, projekce, selekce.
Vzniká při každém SELECT/UPDATE/DELETE dotazu.
Cachování: Může se vyplatit, pokud danou operaci provádí hodně uživatelů najednou *(např. načtení článků ze zpravodajského serveru)*.

11. Jaká je základní strategie pro tvorbu prováděcího plánu? Jsou situace, kdy se vyplatí spíše full-table scan přístup namísto index-based? Případně uveďte.

Vytvoří se více prováděcích plánů, pro každý se spočte jeho cena (cost-based optimalizace) a vybere se plán s nejlepší cenou.
Ano, může se to vyplatit, například pokud mám dotaz WHERE empid > 150 a v tabulce jsou rovnoměrně rozmístěné hodnoty od 1 do 10000 *(skákání po blocích přes indexy bude pomalejší, než sekvenčně přečíst celou tabulku)*.

12. Operace řazení, v jakých situacích se používá, jaké jsou parametry pro odhad ceny řazení.

Např. při sort-merge joinu, použití DISTINCT, ORDER BY, množinových operacích, > nebo <.
Řazení provádíme takzvaným **multi-run** sortem *(všechna data se do memory bloku nevejdou)*.
Cenu odhadujeme podle počtu hodnot a množství dostupných paměťových bloků a dostupnosti speciálních struktur, jako je třeba prioritní fronta.

13. Postup při ladění výkonu DB serveru (jak zjistíme co vázne, jak zvolíme SQL dotazy pro ladění?

Nejprve najdeme v **logu** nejčastější operace, které trvají nejdéle (pravidlo 90:10 nebo 80:20). *(nevyplatí se optimalizovat komplexní dlouhý dotaz, který ale provádíme jen jednou za týden)* Analyzujeme prováděcí plán (EXPLAIN PLAN) a systémové statistiky, můžeme vyřešit přidáním indexu, pokud je třeba.

1. Vysvětlete rozdíly mezi OLTP a OLAP databází.

OLTP: online transaction processing: současně probíhá mnoho transakcí (čtení, zápis), které jsou krátké, real-time, dotazy jsou stejné, liší se jen v parametrech; e-shop, většina systémů
OLAP: online analytical processing: probíhá hodně transakcí nárazově, transakce jsou dlouhé, používá předpočítané hodnoty; datové sklady

2. Vysvětlete, případně uveďte na příkladech hlavní přínos objektově relačních databázových systémů oproti čistě relačním.

ORDBMS rozšiřují klasické relační systémy o objektové prvky *(práce s objekty, uživatelsky definované datové typy)*.
Výhoda: zpracování komplexních objektů, rekurzivní struktury, abstraktní datové typy, API do objektově orientovaných jazyků
Nevýhoda: pomalejší zpracování OLTP, dostupnost, přístup, náročnost, rozšířenost

3. Vysvětlete co je reference na objekt (typ REF) v objektově-relačních databázích. Jaký je rozdíl mezi referencí na objekt a cizím klíčem?

Reference je reference přímo na **řádek** (konkrétní záznam), můžeme skrze ni aktualizovat, upravit objekt, nebo ji změnit.
Cizí klíč obsahuje pouze hodnotu atributu, pro získání konkrétního záznamu musíme provést dodatečný dotaz.

4. Vysvětlete rozdíl mezi relační tabulkou obsahující uživatelem definovaný datový typ a objektovou tabulkou.

Objektová tabulka obsahuje pouze objekty, zatímco **relační tabulka** s uživatelsky definovaným datovým typem může obsahovat i další data. Uživatelsky definované datové typy jsou typicky vnořené tabulky, objekty a kombinace), objektová tabulka obsahuje pouze objekty.

5. V jakém jsou vztahu objektově-relační databázový stroj a ORM (object-relational mapping) technologie? (co to řeší, kdy je co vhodné)

ORDBMS: rozšiřuje relační model o objektové prvky, ukládá objekty přímo do databáze
ORM: mapuje objekty na klasické relační tabulky, ukládá v relační DB (obyčejné tabulky)
ORDBMS je **vhodné** pro efektivní práci s objekty přímo v databázi, ORM pokud chceme pracovat s objekty pouze v kódu v programovacím jazyce.

1. Uveďte a vysvětlete CAP theorém.

CAP: Týká se distribuovaných systémů, systém může mít pouze 2 ze 3 vlastností:
- **C**onsistency: po zápisu musí všechny uzly vidět stejná data *(atomické operace)*
- **A**vailability: každý uzel umí odpovědět na každý dotaz
- **P**artition tolerance: v případě výpadku uzlu je systém schopný dále pracovat

2. Vysvětlete rozdíly mezi koncepcí ACID a BASE.

ACID: systémy zpracovávající transakce, důležitá je **konzistence**, CA v rámci CAP teorému
- **A**tomicita: transakce se provede celá nebo vůbec
- **C**onsistency: po provedení transakce je DB v konzistentním stavu
- **I**ndependence: transakce se provádí nezávisle na sobě
- **D**urability: po provedení je výsledek transakce perzistentně uložen

BASE: distribuované systémy, důležitá je **dostupnost**, AP v rámci CAP teorému
- **B**asicaly **A**vailable: systém jako celek je neustále dostupný
- **S**oft-state: systém není deterministický, po zápisu můžu číst nějakou dobu starou hodnotu
- **E**ventually Consistent: za nějaký čas *(v řádu milisekund)* bude systém konzistentní

3. Co je to horizontální a co vertikální škálování databáze a jak souvisí s CAP?

horizontální škálování: přidávám další servery, distribuované systémy, typicky AP, cenově výhodnější, ale musím řešit distribuci dat a synchornizaci
vertikální škálování: přidávám výpočetní sílu, typicky CA, silná konzistence, má svoje limity (cenové, vendor lock-in, nelze škálovat donekonečna)

4. Jak lze použít CAP theorém ke klasifikaci databázových strojů? Uveďte příklady databázových strojů, které znáte a pokuste se je klasifikovat na základě CAP theorému.

CA: zachovávají ACID vlastnosti, klasické RDBMS *(MySQL, PostgreSQL)*
CP: hlavní je **konzistence**, distribuované zamykání *(MongoDB)*
AP: hlavní je **dostupnost**, BASE *(Cassandra, RiakKV, DNS)*

5. Jaký je rozdíl mezi replikací a technikou sharding? Jsou to techniky, které se vzájemně vylučují nebo se mohou doplňovat?

Replikace: mám více kopií stejných dat na několik serverů
Sharding: jedna data mám rozdělena mezi více serverů *(např. dělím kolekce v MongoDB)*
Tyto techniky lze spolu kombinovat, typicky se tak v NoSQL systémech děje.
Klasicky má každá část dat určité master/slave uzly, které řídí zápis/čtení těchto dat.

6. Co je to silná a slabá konsistence v NoSQL databázích? Jak souvisí s CAP?

Silná konzistence: ACID, všechny změny se **hned** propíšou, maximalizace dostupnosti
Slabá konzistence: BASE, při čtení nemusí každý vždy vidět správně zapsaná všechna data

7. Vysvětlete, co je "quorum" a jak se používá k zajištění silné či slabé konzistence?

Quorum: počet uzlů, které musí zapsat/přečíst data, aby byl požadavek potvrzen.
Silná konzistence = všechny uzly musí zapsat/přečíst, slabá = typicky stačí více než polovina

8. Jak jsou charakterizována BigData (3V+)?
- **V**olume: objem dat je obrovský (Zetabyty)
- **V**elocity: rychlost, kterou data narůstají je obrovská (IoT, sociální sítě)
- **V**ariety: různorodost (data nemají pevnou strukturu a schéma)
- Veracity: nejistota kvůli nekonzistenci, neúplnosti, Value (hodnota), Validity (platnost)

1. Uveďte podstatné rozdíly (výhody a nevýhody) relační a dokumentové databáze.

Relační databáze: instance, databáze, tabulky, řádky, SQL, joiny, normální formy, atomická data, malá redundance, transakce, konzistence, ACID
Dokumentová databáze: instance, dokumenty, typicky JSON/XML identifikován klíčem, bez schématu, rychlejší vytvoření, umožní jen CRUD na základě klíče dokumentu
- využití: velké množství dokumentů s **podobným** schématem (CMS, blogy, event logging)
- nevýhody: množinové operace s dokumenty, agregace

2. Uveďte podstatné rozdíly (výhody a nevýhody) relační a XML-nativní databáze.

XML-nativní databáze: instance, XML dokument identifikován klíčem, XPath, XQuery, stromová struktura, vhodný pro menší a středně velké dokumenty, volitelné schéma
- nevýhody: zpracování velkého množství dat, silná konzistence, silně propojená data

3. Uveďte podstatné rozdíly (výhody a nevýhody) relační a key-value databáze.

Key-value databáze: hash tabulka klíč-hodnota, snadno škálovatelná, CRUD (key, value)
- využití: session data, profily, preference, košík
- nevýhody: komplexní vyhledávání, vyhledávání podle obsahu, relace

4. Uveďte podstatné rozdíly (výhody a nevýhody) relační a grafové databáze.

Grafová databáze: instance, property multigrafy (hrany i uzly můžou mít vlastnosti), přidávání a odebírání uzlu a hrany, aplikace pro grafové algoritmy a podgrafy
- využití: doporučovací systémy, sociální sítě, genealogické či lingvistické stromy
- nevýhody: hodně dávkových operací, moc velké grafy

5. Uveďte podstatné rozdíly (výhody a nevýhody) relační a sloupcové (wide-column) databáze.

Wide-column databáze: instance, column family *(tabulka)*, row *(kolekce sloupců)* s unikátním klíčem, column *(jméno + hodnota – skalár/množina/mapa)*, dotazování podle hodnoty klíče
- využití: záznam událostí, CMS, blogy; nevýhody: neumí JOIN, agregace a pokročilé operace
- MapReduce, Hadoop, paralelní zpracování, responzivita

6. Uveďte výhody a nevýhody přístupů schema-free a schema aware databází.

Schema free: flexibilní, snadné použití a údržba **ALE** nevíme, co je v DB, duplicitní data *(typicky nejsou ani v 1. NF)*, chybí pokročilé optimalizace dotazů a kontrola integrity dat
Schema aware: víme, co je v DB uloženo, umožňuje pokročilé indexování, optimalizace, dokážeme získat obsah databáze **ALE** musíme vytvořit a udržovat schéma (menší flexibilita)

1. Vysvětlete koncepci databázového stroje MongoDB. Uveďte jeho silné stránky a uveďte příklady, kdy je jeho použití vhodné a kdy je naopak nevhodné.

Popis: Dokumentově-orientovaná databáze, hlavní jednotka je dokument s unikátním klíčem. Ukládá se jako binární JSON (BSON), zápis dat a dotazování probíhá pomocí JSON.
Dotazy: create / update / remove / get na dokument *(organizovaný v kolekcích)*.
Schéma se nekontroluje, každý dokument má _id (ObjectId), pomocí něj lze referencovat
Využití: vysoká dostupnost, eventuální konzistence, blogy, CMS, event logging
Nevýhody: redundance dat, konzistence, transakce, joiny, indexování, agregace

2. Vysvětlete koncepci databázového stroje Cassandra. Uveďte jeho silné stránky a uveďte příklady, kde je jeho použití vhodné a kdy je naopak nevhodné.

Popis: wide-column databáze, datový model jsou keyspace => column family (kolekce podobných řádků) => row (kolekce sloupců s unikátním klíčem) => column (name-value)
Hodnota sloupce může být null, atomická hodnota, tuple, kolekce.
Využití: práce s velkým množstvím dat, hodně zápisů, škálovatelnost; záznam událostí
Nevýhody: neumí join, agregaci ani komplexní dotazy, transakční zpracování, neumí ACID

3. Vysvětlete koncepci databázového stroje Neo4j. Uveďte jeho silné stránky a uveďte příklady, kdy je jeho použití vhodné a kdy je naopak nevhodné.

Popis: grafová databáze, databázový model jsou property grafy (orientovaný graf, jak uzly, tak hrany mají vlastnosti + TAG = label, který se dá přiřadit)
Operace: vložení/odebrání uzlu nebo hrany
Využití: doporučovací systémy, sociální sítě, grafové algoritmy *(hledání nejkratších cest…)*
Nevýhody: hodně dávkových operací, moc velké grafy

4. Uveďte koncepci databázového stroje RiakKV. Uveďte jeho silné stránky a uveďte příklady, kdy je jeho použití vhodné a kdy je naopak nevhodné.

Popis: key-value úložiště, přístup k datům je pouze přes klíč *(e-mail, login, autogenerated, ne AUTO INCREMENT !)*, hodnota je black-box
Datový model: bucket (logická kolekce key-value objektů), lze seskupovat do bucket types
Vlastnosti: expirace key-value párů *(session na webu)*, linkování párů, kolekce hodnot
Využití: session data, profily, preference, košík ; výhody: sharding, replikace, dostupnost
Nevýhody: vyhledávání podle obsahu, komplexní vyhledávání, relace

5. SAP HANA- Uveďte zajímavé (specifické) rysy, silné a slabé stránky, vhodné použití.

Popis: in-memory databáze založená na sloupcích, vysoká dostupnost, škálovatelná
Využití: mapování záznamů časových řad, dat ze senzorů do strukturovaných dat
Slabé stránky: škálování je ve vývoji, neumí recovery z chyby, databáze se musí vejít do DRAM *(hodí se jen pro menší a střední data)*, funguje jen na certifikovaných zařízeních

1. Krátce popište, případně vysvětlete na vhodných příkladech dotazovací jazyk Cypher.

Slouží k dotazování nad grafovou databází Neo4j. Matchuje podgrafy, dá se libovolně řetězit.

```
MATCH (m:MOVIE)-[:PLAY]->(a:ACTOR)
WITH m, SIZE([q = (m)-[:PLAY]->(:ACTOR) | q]) AS actors
  WHERE m.title = "Štěstí"
RETURN DISTINCT a.name, a.year
  ORDER BY a.year
```

Umožňuje řešit orientovanost hran, používat labely, označovat proměnné a bindovat je. Umí pracovat i s délkou cest a cestami [:PLAY *1..3], nebo ukládat relace a pracovat s nimi.

2. Krátce popište, případně vysvětlete na vhodných příkladech dotazovací jazyk XQuery.

Slouží k dotazování v XML-native databázích, pracuje nad stromovou strukturou XML. Nadstavba XPath, umožňuje složitější konstrukce.

XPath: pracuje s osami (self, child, descendant, attribute) a odpovídajícími zkratkami, umožňuje pracovat s predikáty, porovnávat
XQuery: rozšiřuje, obsahuje FLWOR (for, let, where, order by, return)

```
for $m in //movie
   let $r := $m/@rating
   where $r >= 75
   order by $m/@year
   return $m/title/text()
```

Umožňuje konstrukci vlastních dat (return <movie><name>{ $m }</name></movie>)

3. Krátce popište, případně vysvětlete na vhodných příkladech dotazovací jazyk MongoDB.

Slouží k dotazování v MongoDB databázi, pracuje nad JSON dokumenty v MongoDB. Základní operace je db.**kolekce**.find(dotaz, projekce).sort(řazení).pretty()

```
db.movies.find(
  { year: { $gt: 2005 } },
  { _id: false, title: true }
).sort({ title: -1 })
```

Umí pracovat s bool dotazy, poli, při dotazování záleží na pořadí, lze pracovat s vnořenými properties pomocí "actor.firstname".

1. Charakterizujte rozdíly mezi tzv. micro a complex benchmarkem v databázích.

Micro-benchmark: zaměřuje se na jednu konkrétní operaci *(čtení, vkládání)*. Výhodou je zúžení na část systému, kterou chceme zrychlit / prozkoumat. Tyto testy jsou rychlé.
Complex benchmark: se zaměřuje na systém jako celek, snaží se systém otestovat v reálných podmínkách jako celek

2. Co je TPC a jak souvisí s databázovými benchmarky?

Transaction **P**rocessing Performance **C**ouncil, spojuje výrobce databázových systémů a hardwaru. **Cíl**: definuje různé benchmark testy databází tak, aby byly akceptovány komunitou.
OLTP benchmarky: TPC-E (burza), TPC-C (e-shop); **OLAP** benchmarky: TPC-H (datový sklad)

3. Vysvětlete princip benchmarku TPC-C. Co je výstupem benchmarku?

TPC-C: simulace databáze e-shopu (9 tabulek), simuluje klienty, kteří vytváří session, přidávají věci do košíku, aktualizují účty, platí, sledují stav zboží, monitoruje se stav skladu…
Metrika: TPM-C *(počet transakcí za minutu)* = kolik systém při daném zatížení zvládne zpracovat nových objednávek za minutu
Další metriky: kolik výkonu stojí nová objednávka, jak se systém umí naškálovat

4. Vysvětlete princip benchmarku TPC-E. Co je výstupem benchmarku?

TPC-E: simulace systému pro obchodování na burze, 25 tabulek, složité vazby mezi tabulkami. Oproti TPC-C obsahuje reálnější data (zákazníci s uvěřitelnými jmény, adresami), provádí se hledání podle jmen a adres.
Metrika: TPS-E *(transakcí za sekundu)* = počet trade result transakcí za časovou periodu
Další metriky: kolik výkonu stojí jedna trade result transakce

5. Vysvětlete princip benchmarku TPC-H. Co je výstupem benchmarku?

TPC-H: testuje analytické zpracování v datovém skladu, nad kterým děláme analýzy
Netestuje DML operace *(sklady se mění jednou za den)*, ale **analýzy** (většina zátěže).
Metrika: QPH-H *(composite query per hour)* = počet zpracovaných komplexních dotazů za hodinu
Další metriky: kolik výkonu stojí zpracování jednoho takového komplexního dotazu