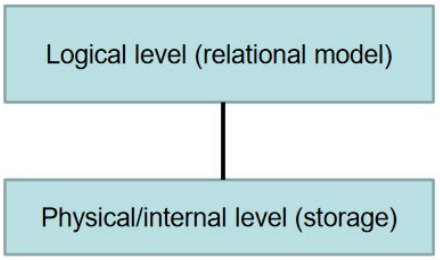
Database Advanced Exam - Összefoglaló

A doksit a diákból és korábbi kisZH kérdésekből dobtam össze. Felelősséget nyilván nem vállalok blabla

Database management, advanced indexes

Database architecture:



Block: A contiguous sequence of sectors from a single track

- Data is transferred between disk and main memory blocks
- Sizes (4 / 8 / 16 / 32 kB)
    - Smaller blocks: more transfers from disk
    - Larger blocks: more space wasted due to partially filled blocks

Index mechanisms used to speed up access to desired data.
Search key: attribute (or set of attributes) used to look up records in a file.
An **index (file)** consists of **index entries** of the form:

Two basic kind of indices:

- **Ordered indices**: search keys are stored in sorted order in some kind of a search tree
- **Hash indices**: search keys are distributed uniformly across buckets using a hash function

A **B+ tree** is an N-ary tree with a variable but often large number of children per node. A B+ tree consists of a root, internal nodes and leaves. The root may be either a leaf or a node with two or more children.

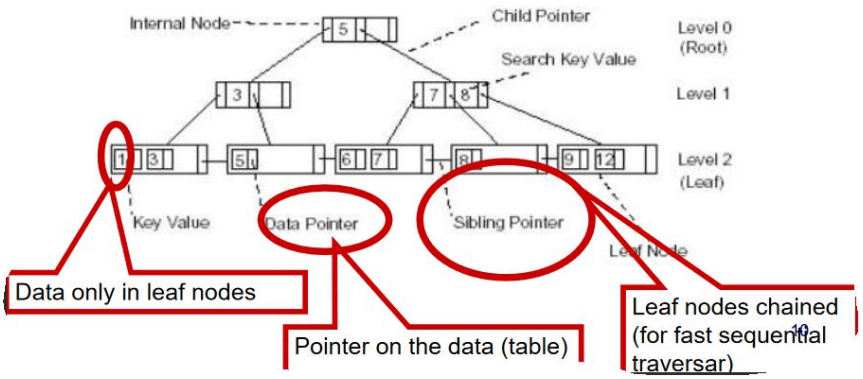
A fa egy gyökérelemmel (root) rendelkezik, ami alatt több szinten köztes csomópontok találhatók, legalul pedig levelek (leaf) helyezkednek el. Az ábrán egyes csomópontok 1-1 blokkot jelölnek. A gyökér és a köztes csomópontok azonos felépítésűek, kulcs-mutató párokból állnak. A levelekben kulcs, rowID párok találhatók, alapesetben a tábla minden sorára egy ilyen pointer mutat.

Typical size of index

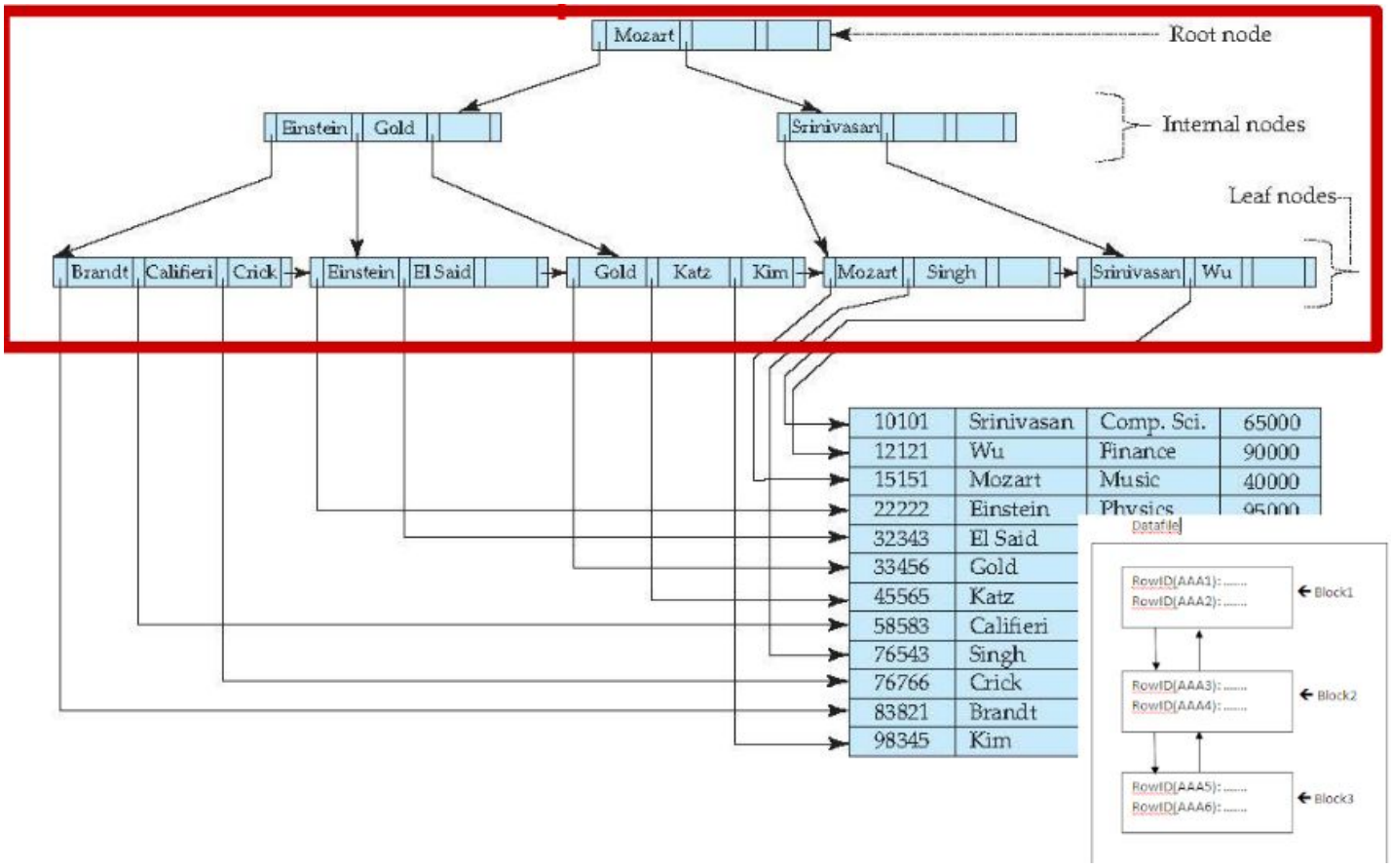
- Node ~ block on disk
- Number of children of a node
- Depth of an index

Parts and other properties of a B+ tree

- Root node, internal nodes, leaf nodes
- Key value, Data pointer, Sibling pointer



Example of B+ tree



When to use indexes?

- Useful for sorted results
- Join conditions
- Query conditions
    - Exact match / range queries
    - Selectivity (Index is useful for a query condition with at most ~5% selectivity) **(Clustering factor)**

B+ tree index files

- **Advantages of B+ tree index files:**
    - Automatically reorganizes itself with small, local, changes, in the face of insertions and deletions
    - Reorganization of entire file is not required to maintain performance
- **Disadvantages of B+ trees:**
    - Extra insertion and deletion overhead
    - Space overhead (size of index comparable to size of table)

Composite search keys are search keys containing more than one attribute.

We say that an **index is covering** for a specific query if the index contains all the needed attributes - meaning the query can be answered using the index alone.

The **cost of a search** in B+ tree index depends in the **depth of the tree**, which in turn can be estimated by **taking logarithm** of the number of values.

Emígy lehet indexet csinálni:

```
CREATE INDEX index_name
ON table_name (column1, column2, ...);
```

Van néhány olyan alkalmazási terület, melyet a szokásos B+ fa index nem támogat kellőképpen.

- Földrajzi lekérdezések -> Az adatok nem rendezhetőek
- DWM sok rekordot érintő (nem szelektív) lekérdezések.

Ezekre persze van alternatíva:

- R-fa a földrajzi lekérdezésekhez
- Bitmap-index, max. view

Join execution, database optimization

In relational databases we use a large number of joins, however it is a very expensive operation. There are several ways to execute it.

Nested loop join

```
Compute R ⋈ S on A:
 for r in R:
  for s in S:
   if r[A] == s[A]:
    yield (r,s)
```

1. Loop over the tuples in R
2. For every tuple in R loop over all tuples in S
3. Check against join conditions
4. Write out (to page, then when full, to disk)

Cost: $P(R) + T(R) * P(S) + OUT$

Pros: any join condition
Cons: expensive (better if tables fit in memory)

Block nested loop join

```
Compute R ⋈ S on A:
 for each B-1 pages pr of R:
  for page ps of S:
   for each tuple r in pr:
    for each tuple s in ps:
     if r[A] == s[A]:
      yield (r,s)
```

1. Load in B-1 pages of R at a time (leaving 1 page each free for S & output)
2. For each B-1 page segment of R load each page of S
3. Check against the join conditions
4. Write out

Cost: $P(R) + \frac{P(R)}{B-1} * P(S) + OUT$

BNLJ vs NLJ
In BNLJ by loading larger chunks of R, we minimize the number of full disk reads of S. BNLJ is significantly faster.

Sort-Merge-Join
Sort relations by join attribute.
Join operators: <, <=, >, >=

Simple Hash Join

- Partition relation R in $R_1, R_2, ..., R_p$ with Hash-function h so that for all tuples $r \in R_i$ $h(r.A) \in H_i$.
- Scan relation S, apply for each tuple $s \in S$ the hash function h. Is $h(s.B)$ in H_i, search there for fitting r.
- Size of hash table > Available RAM
    - Multiple scans, can be very slow
    - Small increase in data cause large increase in run time
- Pro: fast
- Cons: memory requirements, join operator: only "="

Query execution

- SQL is very high level and declarative
- SQL query is translated by the query processor into a low level program: the execution plan
- The execution plan is a program that can be executed to get the answer to the query.

Relational algebra equivalences
Key idea comes from algebraic optimization and laws.

1. Selection

$$\sigma_{c1 \wedge c2 \wedge ... \wedge cn}(R) \equiv \sigma_{c1}(\sigma_{c2}(...(\sigma_{cn}(R))\ ...))$$

2. σ is commutative

$$\sigma_{c1}(\sigma_{c2}((R)) \equiv \sigma_{c2}(\sigma_{c1}((R))$$

3. π - cascades

$$L_1 \subseteq L_2 \subseteq ... \subseteq L_n$$

$$\pi_{L1}(\pi_{L2}\ (...(\pi_{Ln}(R))\ ...)) \equiv \pi_{L1}(R)$$

4. Changing σ and π

If selection only refers to the projected attributes $A_1, ..., A_n$, selection and projection can be exchanged.

$$\sigma_c(\pi_{A_1, ..., A_n}(R)) \equiv \pi_{A_1, ..., A_n}(\sigma_c(R))$$

etc.

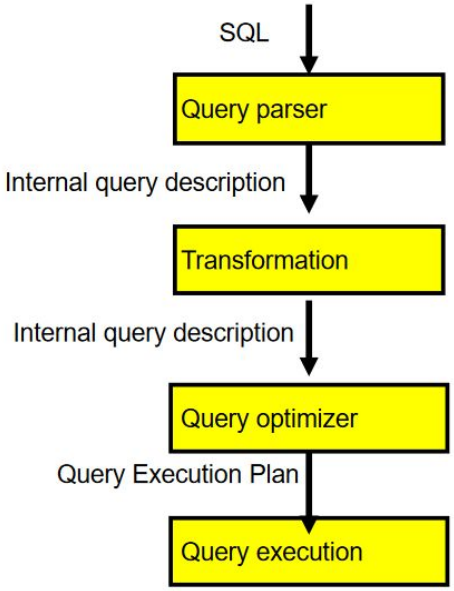
Query optimization
One SQL query - many different execution plans, execution alternatives, and they have different cost.

- Index - using / not using
- Join execution?
- Join order?

Query optimizer: choosing a relatively good execution plan.

Query execution

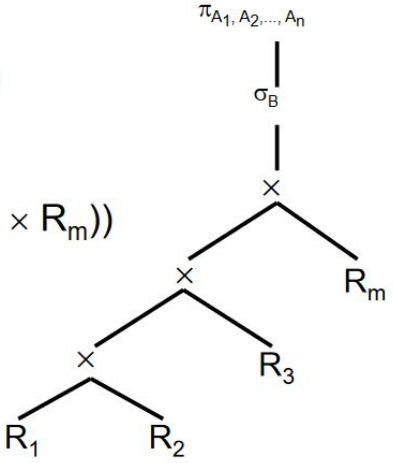
- **Query parser:** Parsing, Semantic check, view expansion
- **Transformation:** standardized description
- **Query optimizer:** Optimisation (cost based): setting up execution plans, estimating their cost, select the best
- **Query execution:** Execute the selected execution plan



Transformation, operator tree

```
select   A1, A2, ..., An
from R1, R2, ..., Rm
where    B
```

$\Rightarrow \pi_{A_1, A_2, ..., A_n} (\sigma_B (R_1 \times R_2 \times ... \times R_m))$



Basic idea:

Execute σ, π early, and the others (eg. join) late, as

- σ, π reduces the volume of data
- And the others often in large intermediate results.

Execution mode

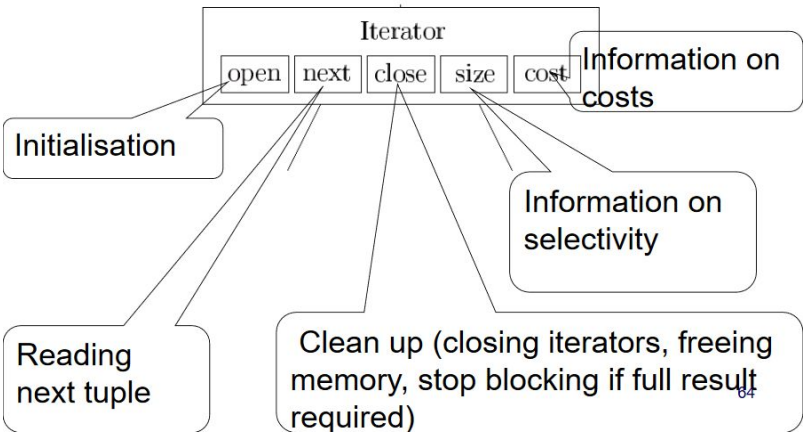
Full calculation: Node fully calculated before next operator

Pipeline: Tuple calculated at one node sent immediately to next node

Pipeline-breaker

A pipeline breaker is an operator that produces the first (correct)output tuple only after all input tuples have been processed.

Unified description of operators



TOP-N Query

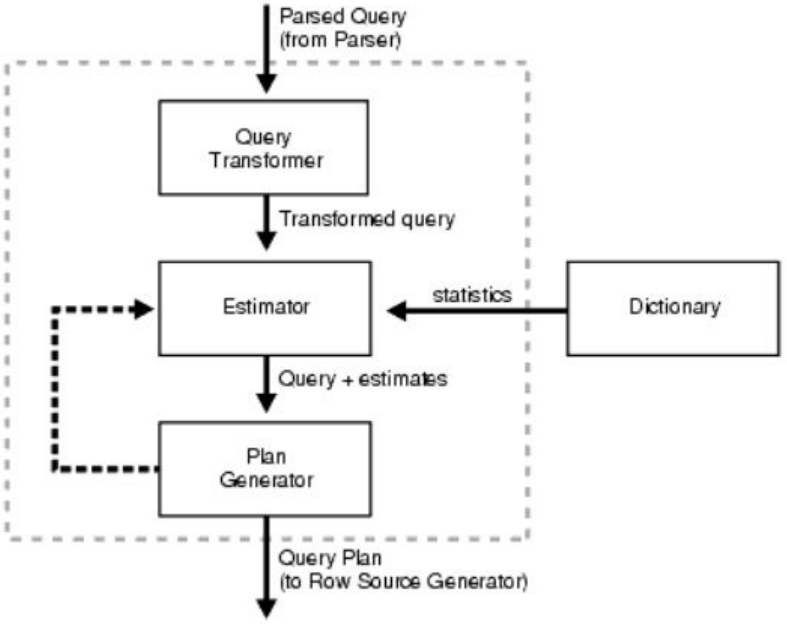
- Only the first N rows of the result are required.
- Execution plan optimized specifically for this (hard decisions, soft decisions)

Cost function

The cost is an estimated value proportional to the expected resource use needed to execute the statement with a particular plan. The optimizer calculates the cost of access paths and join orders based on the estimated computer resources, which includes I/O, CPU, and memory.

- Cost of execution
    - One major factor: disk access, number of blocks
- Size of intermediate results are also important.
- DBMS calculates and stores different statics on the data (Gondolom, ez segítség, de igazából nem teljesen értem)

Components of the Query Optimizer



Korábbi zh-ból:

A költség alapú lekérdezés optimalizálás célja, hogy a **procedurális/deklaratív** (jellegű) SQL nyelven megadott feladat (procedurális) végrehajtását meghatározza. A lekérdezés egy szintaktikai ellenőrzés után egy belső formába az ún. **végrehajtási tervbe (execution plan)** transzformálódik. Ezen különböző transzformációkat lehet végrehajtani melyek során az **eredmény (result)** nem változik de a végrehajtás **sorrendje (order)** igen.

Ökölszabály, hogy a közbülső eredmények **minimálisak (minimal)** legyenek.

Példák a költség alapú lekérdezés optimalizálás során hozandó döntésekre: **(1) index, (2) join végrehajtás, (3) join sorrend**.
Általában pl a **szűrés** és **vetítés** műveleteket érdemes lehetőség szerint korán, míg a **JOIN összekapcsolás** és az **UNIÓ** műveleteket későn végrehajtani.
A lehetséges végrehajtási tervek közül a költségük alapján kerül kiválasztásra végrehajtandó végrehajtási terv.
A költségek becslésének alapja az adatokról vezetett **statisztika**, például a táblákban lévő rekordok **mérete**, egyes attribútumok értékeinek **típusa**.

The goal of the cost based query optimisation is to find an **execution plan** with a relatively low resource requirement for a declarative query given in SQL. After syntax check, the query is transformed into an internal representation, called execution plan. On this several transformations can be done, which change the **execution order**, but do not affect the **result**.
As a rule of thumb, intermediate results shall be **minimal**. Examples of decisions a cost based query optimizer makes: **(1) index, (2) join execution, (3) join order**.

1. The depth of a typical B+ tree based index is around 30.
2. The size of a node in a typical B+ tree based index corresponds to the size of a block on the hdd.
3. The typical size of block on the hdd is around 1kB.
4. A typical (B+ tree based) database index supports well queries with a high selectivity.
5. A B+ tree based index I(A,B) created for the table T(A,B,C) can be used for range queries on attribute A.
6. A B+ tree based index can only be used for exact match queries.
7. A B+ tree based index I(A,B) created for the table T(C,B,A) can be used for range query conditions on attribute A.
8. A B+ tree based index I(A,B) created for the table T(C,B,A) can be used for an exact match query condition on attribute B.
9. A function based join condition is typically executed by the sort-merge join method.
10. The join condition T1.A < T2.B is typically executed by the hash join method.
11. If a table grows by a factor of 10 000, the corresponding B+ tree based index will typically require 10 times more space.
12. A B+ tree based in index supports well queries selecting/aggregating a large number of records (queries typical for data warehouses).
13. If an INNER JOIN is replaced by an RIGHT OUTER JOIN in a query, the number of the records in the result may increase.
14. If an INNER JOIN is replaced by a FULL OUTER JOIN in a query, the number of the records in the result always increases.
15. Schema constraints (e.g. foreign key, not null) make query execution typically more expensive.
16. The joins in a query are executed in exactly the same order as they appear in the SELECT command.
17. During cost-based query optimization, the result of the query may change slightly.

	Igen	Nem
1		X
2	X	
3		X
4	X	
5	X	
6		X
7	X	
8		X
9	X	
10		X
11		X
12		X
13	X	
14		X
15		
16		X
17		X

1. Egy index csak akkor használható, ha pontos értékre keresünk (exact match query).
2. Egy T(A,B,C) táblán létrehozott I(A,B) indexet lehet egy A-ra vonatkozó tartományi lekérdezésre (range query) használni.
3. Egy T(A,B,C) táblán létrehozott I(A,B) indexet lehet használni egy B-re vonatkozó pontos egyezés (exact match) lekérdezésre.
4. A szokásos (B+ fa alapú) indexekkel hatékonyan lehet földrajzi lekérdezéseket támogatni.
5. A szokásos (B+ fa alapú) indexekben a csomópontok gyerekeinek száma ("fanout") 100-as nagyságrendben van.
6. A szokásos (B+ fa alapú) indexekben a csomópontok mérete a HDD-n levő adatblokkok méretéhez igazodik.
7. Ha egy tábla mérete 10 000 szeresre növekedik, a hozzá tartozó (B+ fa alapú) index mérete kb. 100 szorosra fog növekedni.
8. A T1.A < T2.B összekapcsolási feltétel esetén az adatbáziskezelő a hash joint fogja használni.

	Igen	Nem
1		X
2	X	
3		X
4	X	
5	X	
6	X	
7		X
8		X

Object-relational databases, geographical databases

Object-oriented databases ("object databases")

- Use the object-oriented principles and concepts: encapsulation, inheritance, etc.
- 90-es években kezdenek ezzel foglalkozni
- Object-oriented DBMS are not widely used since they don't offer the efficiencies of well-entrenched relational DBMS.

Object-oriented extensions to relational DBMS

- Use the advantages of OOP, yet retain the relation as the fundamental abstraction
- Support more complex data types, not just flat files (Maps, XML, multimedia, etc.)
- The relational model supports very high level queries.

Oracle: User Defined Types

A user defined type or UDT is essentially a class definition with a structure and methods.
Two uses: As the type of an attribute of a relation, as a rowtype that is the type of a relation.

```
SQL> CREATE TABLE states (
  2     state       VARCHAR2(30),
  3     totpop      NUMBER(9),
  4     geom        SDO_GEOMETRY);
```

```
create type emp_t as object (empno number, ename varchar2(10), ...);
create table emp of emp_t;
```

Parts of object-relational DBMS:

- Extensible data types
- Methods / Operations
- Extensible indexing (cost-based query optimization)

Typical use cases:

- Geographical data / Spatial database (Maps) (a legfontosabb) - használt program: PostgreSQL
- Multimedia / complex data types eg. XML - használt program: Oracle

To hold and store regular data it is **not necessarily better to use object-relational databases**. In most cases the "old-fashioned" relational tables are simply better.

Oracle XML DB
Oracle XML DB and XML Developer's Kt enable you to develop high performance applications that process XML content and manage XML stored in the database.

PostgreSQL vs Oracle
A PostgreSQL, más néven Postgres egy relációsadatbázis-kezelő rendszer.
Oracle:

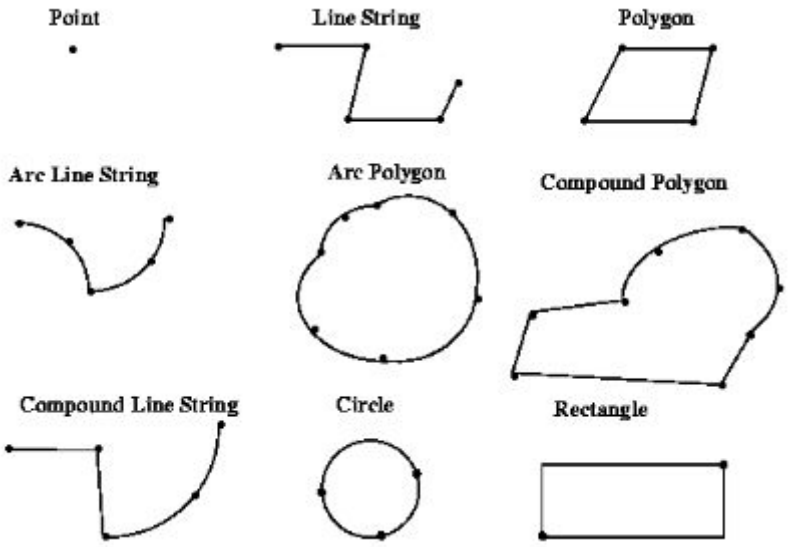
- Define type, create table using type
- Methods
- No inheritance between tables
- No multiple parents
- Smart column types

PostgreSQL

- Table inheritance

PostGIS geographic data types

```
CREATE TABLE testgeog (
 gid serial PRIMARY KEY,
 the_geog geometry( point, 4326 )
);
```



Spatial Reference System, SRID

- World Geodetic System
  - WGS 1984
  - SRID 4326
  - Google, KML
- Hungary: Egységes Országos Vetület (SRID 23700)

Geographic index

- Bounding box
- R-tree
- Query processing (multiple filter sets)

Spatial Database Offerings

- Oracle Spatial
- MS SQL Server

- Geomedia on MS Access
- PostGIS / PostgreSQL

Practical hints

- PostgreSQL + PostGIS (+ pgRouting)
- Visualization: QGIS - Desktop GIS vagy Google
- Data: OpenStreetMap

1. A PL/SQL az SQL olyan procedurális kiterjesztése, amely jól szabványosított.
2. A PL/SQL programkód az adatbáziskezelő szerveren fut le, azzal jól integrált (ebből következően pl. tranzakciókat és a jogosultságkezelést is támogatja)
3. Az objektumorientált adatbáziskezelő rendszerek mára rendkívül elterjetté váltak.
4. Sok relációs adatbáziskezelőként ismert rendszer (pl. Oracle, PostgreSQL) tartalmaznak objektumrelációs kiterjesztéseket, és így valójában objektumrelációs adatbáziskezelők.
5. A B+ fa alapú indexek jól használhatóak földrajzi adatokhoz.
6. Az objektumrelációs adatbáziskezelés jól szabványosított, valamennyi gyártó megvalósítása egységes.
7. Célszerű a objektumrelációs lehetőségeket akár egyszerű üzleti adatokhoz is használni.
8. A Google által használt földrajzi vonatkoztatási rendszer, a World Geodetic System (SRID=4326) közvetlenül és egyszerűen használható távolságmérésre.
9. Egy adattárház tulajdonképpen tranzakciós adatok másolata, kifejezetten lekérdezési, elemzési célra.
10. A B+ fa alapú index jól támogatja az adattárházak átfogó lekérdezéseit.
11. Egy materializált nézet lényegében bármilyen SQL kifejezést tartalmazhat.
12. Az in-memory adatbáziskezelőkben feleslegesség vált az SQL procedurális kiterjesztése, az SAP HANA rendszere is csak (deklaratív) SQL-t támogat.

	Igaz	Hamis
1	X	
2	X	
3		X
4	X	
5		X
6		X
7		X
8	X	
9	X	
10		X
11		X
12		X

JDBC

Mi az a JDBC?

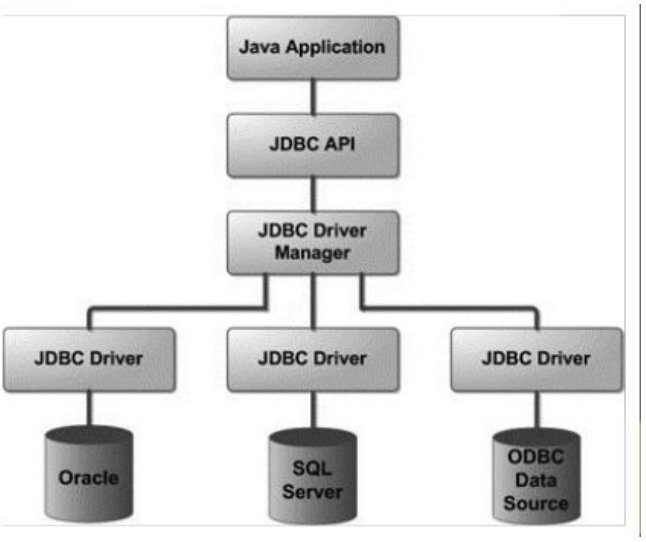
Java Database Connectivity, Java programnyelvhez készül API, adatbázis hozzáférést tesz lehetővé. Képes

- Kapcsolatok létrehozására adatbázisokkal
- Lekérdező / módosító parancsok küldésére
- A lekérdezés eredményeinek feldolgozására

Van ennek egy kedves párja .NET-es Microsoftos világban azaz ODBC (Open Database Connectivity)

JDBC felépítése

- Java alkalmazás (kapcsolat nyitás / zárás, SQL kódok küldése)
- Driver Manager (ez tölti be a JDBC Drivert)
- Driver (az adatbázis-kezelő gyártója adja)
- Data Source (végrehajtja a lekérdezést)



JDBC használati lépések

- Betöltjük a Java programunkba a Drivert
- Csatlakoztatjuk az adatforráshoz
- Végrehajtjuk az SQL utasítást

Részletesen:

- JVM betölti a Driver interfészt megvalósító osztályt és regisztrálja a DriverManagernél a DriverManagertől kérünk egy Connectiont.
- Connectiontől egy Statementet
- A Statementtel végrehajtjuk az SQL utasítást, aminek az eredménye egy ResultSet
- Ez a ResultSet egy iterátor segítségével bejárható
- Végén lezárjuk a kapcsolatot.

JDBC driver betöltés módjai

- DataSource
  - DataSource interfész segítségével az adatforrás használata jobban optimalizálható
  - Adatforrás URL-jét nem szükséges fixen kódolnunk, JNDI-t elég ismerni -> hordozhatóság.
- DataManager
  - Mindent kell ismerni a kódban (host, port username, password, driver class)

Mi az a JDBC driver? Milyen típusai vannak? (Melyik a leggyakoribb típus, amely egyben az Oracle thin driver típusa is?)

Kliensoldali adapter, amely a Java kérései átalakítja az adatbázis-szerver által értelmezhető formára.

- Type 1 - JDBC-ODBC híd
- Type 2 - natív-api driver
- Type 3 - hálózati protokoll (middleware)
- Type 4 - native protocol driver, pure java
  - Install JVM on client side, better performance. different vendors different protocols.

Statement használata

- **Statement:** Segítségével SQL utasításokat hajtunk végre, melyek ResultSet
  - executeQuery: a paraméterében megadott SQL lekérdezést végrehajtja, majd az eredménytáblával (ResultSet) tér vissza.
  - executeUpdate: a paraméterben megadott SQL utasítást végrehajtja, majd a módosított sorok számával tér vissza.
  - execute: előzőek általánosítása. True az értéke, ha az eredmény ResultSet típusú. Ekkor ez egy függvénnyel lekérhető.

```
Statement stat1 = myCon.createStatement ();
```

```
Stat1.executeQuery("SELECT * FROM test");
```

- **PreparedStatement**: Paraméterezhető SQL lekérdezés
  - Előnyei:
    - Cachelődik
    - Lekérdezési terv csak egyszer készül el
    - SQL Injection ellen véd

```
PreparedStatement stat2 = myCon.prepareStatement ("SELECT beer, price FROM
Sells WHERE bar= ?");
```

- **CallableStatement**: Eljárás- és függvényhíváshoz

```
CallableStatement stat3 = myCon.prepareCall ("{call JoeMenu(?, ?)}");
```

ResultSet

A lekérdezés eredményeit tartalmazza soronként, úgy viselkedik, mint egy kurzor/iterátor.

- next() - tel kövi sorba megy és true-t ad, ha még van elem, ha nincs false
- getInt(i), getString(i) - információ kiszedése, i sorszám / név (tábla oszlopnév igazából)
- previous(), last(), first() - a next()-hez hasonlóan

Egy utasítás végrehajtása akkor zárul le, ha az összes visszaadott eredménytábla fel lett dolgozva (minden sora kiolvasásra került). Az utasítás végrehajtása manuálisan is lezárható a Statement.close metódussal.

```
ResultSet menu = stat1.executeQuery("SELECT beer, price FROM Sells WHERE bar =
'Joe''s Bar');
```

Tranzakció kezelés

- Intermediate az alapértelmezés
- Autocommit van alapértelmezésben, amit át lehet természetesen állítani és használni a commit(), rollback() metódusokat igény szerint
- CLOSE() fontos lezárni a kapcsolatot már csak a tranzakció kezelések miatt is.

Mi az az SQL(code) injection?

Az SQL injection egy platformfüggetlen hackelési eszköz, mely speciális db lekérdezést küld a kiszolgálónak, hogy hozzáférjen a db-hez. Erre egy koncepciós hiba kínál lehetőséget.
$query = “SELECT user FROM tbl_user WHERE name = ‘$name’”;
Hacker inputs: admin’ OR 1=1 --

Miért használnak connection poolt?

To enhance the performance of executing commands on a db. After a connection is created, it is placed in the pool and it is used again so that a new connection doesn't have to be established.

ORM, Hibernate

MI az Object-Relational Mapping (ORM)?

Objektum relációs leképezés: Az ORM egy programozási technika, ami objektum orientált rendszert hivatott leképezni egy nem kompatibilis rendszerbe (pl. relációs adatbázis). A cél, hogy az objektumok tulajdonságait és kapcsolatait megőrizzük.

Ez egy kétirányú leképezés az objektumorientált világ és a relációs modell között. A kettő között az alábbiak szerint köthető megfeleltetés:

- Osztály - Tábla
- Példány - Sor

Mi az a Hibernate?

- A Hibernate egy teljesen Java alapú framework és egy ORM, amely lehetővé teszi az objektumrelációs leképezést: POJO-k relációs adatbázishoz való kapcsolását.
- Nagyon hatékony leképezést hoz létre az objektumok és az adatbázis táblák közt.
- A fejlesztés sokkal egyszerűbb és gyorsabb.

Hibernate kommunikációs egységek

- **SessionFactory**: Szálbiztos objektum Session létrehozása. Az appok az ezt az interfészt implementáló osztály egyetlen példányát használják.
- **Session**: Ennek az interfésznek az implementációi biztosítják az entitásokon való műveletek végzését. A kliens és adatbázis közötti kommunikációt reprezentálja. A komm. során Transaction objektumokat hoznak létre, ezek reprezentálják az elvégzett munkát. Nem szálbiztos.
- **Perzisztens objektumok**: Rövid életű objektumok, egy sessionhöz tartoznak. Ha a session véget ér, akkor elérhetővé válnak más objektumok számára is.

Hibernate felépítése

- Java osztály
- Relációs adatbázisbeli táblák
- Descriptor (konverziós szabályok definiálása):
    - XML
    - Annotáció

Hibernate előnyei

- Objektumokat használ táblák helyett, objektum perzisztencia
- Rövidebb fejlesztési idő
- Könnyen karbantartható
- Cachelés (lazy vagy eager)
- Version property (optimistic locking)
- Adatbáziskezelőtől független fejlesztés
- Connection pooling

Hátrány: nyilván lassabb valamivel

Konfigurációs fájl

- hibernate.cfg.xml
- Projektben létrehozod új fájlként
- Aztán ebben minden féle csúnya dolog van, szerver cím, portok, username, jeszó, stb.

Annotációk

Annotációk ezek a kis cucclik javaban amik előtt kukac (értsd: @) van (pl. @Override). Ezek a programkód elemeihez rendelhetők (Pl. csomagokhoz, típusokhoz, metódusokhoz, attribútumokhoz, konstruktorokhoz, lokális változókhoz), plusz információt hordoznak a Java fordító ill. speciális eszközök számára.
Ilyen volt Hibernateben pl amikor egy osztályt próbáltunk a táblához megfeleltetni:

```
@Entity(name="User_table")
public class User {

  @Id @GeneratedValue  // primary key és generáljuk a keyt
  int userId;

  @Column(name="User_Name") // adatbázisban oszlop neve
  String userName;

  ....stb
}
```

Kapcsolatok

- @OneToOne
  - 1-1 kapcsolat, Join Column-al történik
- @ManyToOne
  - N-1 az N oldali netitásban, Join Column (kapcsoló mező)
- @OneToMany
  - 1-N az 1 oldali entitásban, Join Table. Alapból ez az adattag a kapcsolódó entitás típusából készített kollekciós / lista lesz, de érdemesebb az N oldali entitás táblájába illesztett kapcsoló mezővel megvalósítani
- @ManyToMany
  - N-M, Join Table, adattag típusa itt is a kapcsolódóból készített kollekció, de lehet érdemes köztes entitást használni

MIlyen két lehetőség van a kapcsolódó objektumok betöltésére?

- **Lusta(Lazy)**: kollekció tartalma akkor inicializálódik, amikor először hivatkozunk rá. / a kapcsolatot csak akkor töltjük be, ha azt külön kérjük
- **Mohó (Eager)**: Kollekció betöltése után rögtön van hivatkozás rá. / az objektum betöltése során betölti a hozzá tartozó kapcsolatot

Használna ORM-et (Hibernate-et) egy tranzakciós alkalmazás esetén? Egy adattárházban? (Gondolkodós feladat, próbálja megindokolni)
Hibernate főleg kisebb adatok 1-1 kapcsolat esetén hasznos, viszont nagyobb adattárházak esetén a nagy adatállomány miatt nem jó.
Ilyenkor jobb a JDBC.

Data warehousing

Rövid definíció: A copy of transaction data, specifically arranged, grouped from operational data sources, to support **business decisions**, reporting etc.

Definíció 2 *(ezt nem kell szerintem tudni, csak segít, hogy értsd MÉG JOBBAN): A data warehouse is a storage architecture designed to hold data extracted from transaction systems, operational data stores and external sources. The warehouse then combines that data in an aggregate, summary form suitable for enterprise-wide data analysis and reporting for predefined business needs.*

The five components of a data warehouse are:

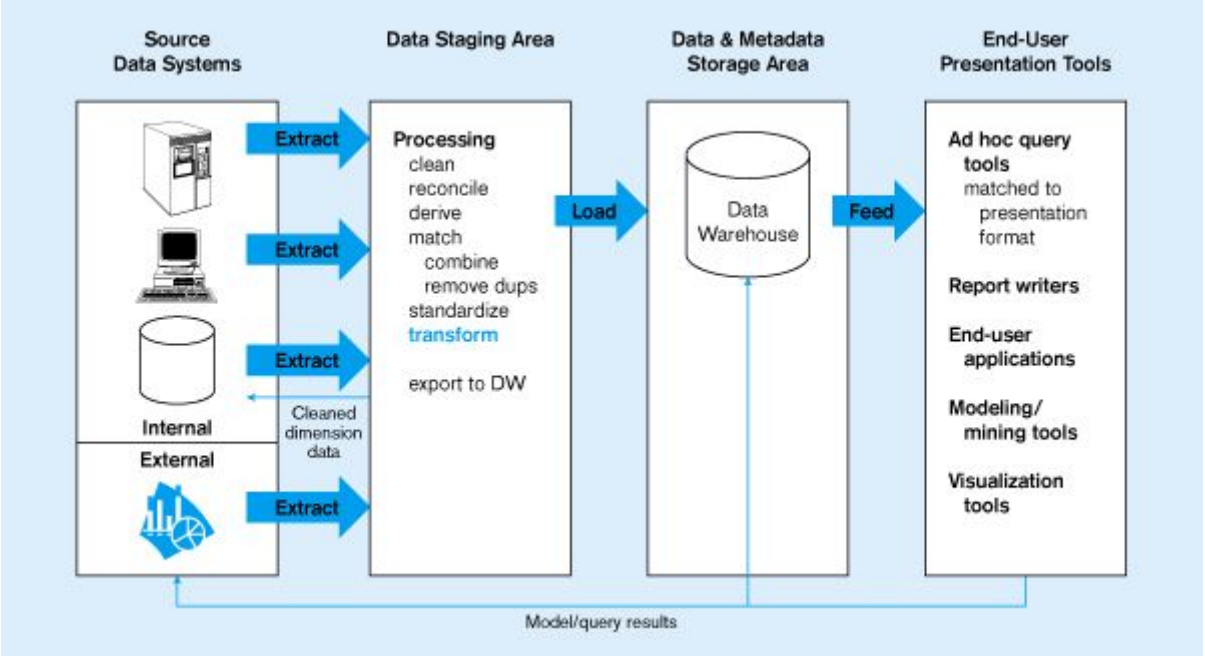
- Production data sources
- Data extraction and conversion
- The data warehouse database management system

- Data warehouse administration
- Business intelligence (BI) tools

Definíció 3 *(hasonlóan def2-höz, nem hiszem hogy kell):*

- ***Subject-oriented****: a data warehouse can be used to analyze a particular subject area. For example "sales" can be a particular subject.*
- ***Integrated****: a data warehouse integrates data from multiple data sources. For example source A and source B my have different ways of identifying a product, but in a data warehouse, there will be only a single way of identifying a product.*

Szokásos architektúra:



Az ETL betűszó jelentése: Extract, Transform, Load

Data Reconciliation

Typical operational data is:

- Transient - not historical
- Restricted in scope - not comprehensive
- Sometimes poor quality - inconsistencies and errors

After ETL, data should be:

- Detailed - not summarized yet
- Historical - periodic
- Comprehensive - enterprise-wide perspective
- Quality controlled - accurate with full integrity

The ETL process

- Capture
- Scrub or data cleansing
- Transform
- Load and index

ETL tools

- Declarative rules (what)
    - Mappings and transformations defined graphically
- Actual implementation (how)

- Handling errors, performance, auditing, method (also change data capture)

Querying Tools

- SQL is not an analytical language
- SQL-99: some data warehousing extension, but still not a full featured data warehouse querying and analysis tool
- Extensions: OLAP

OLAP

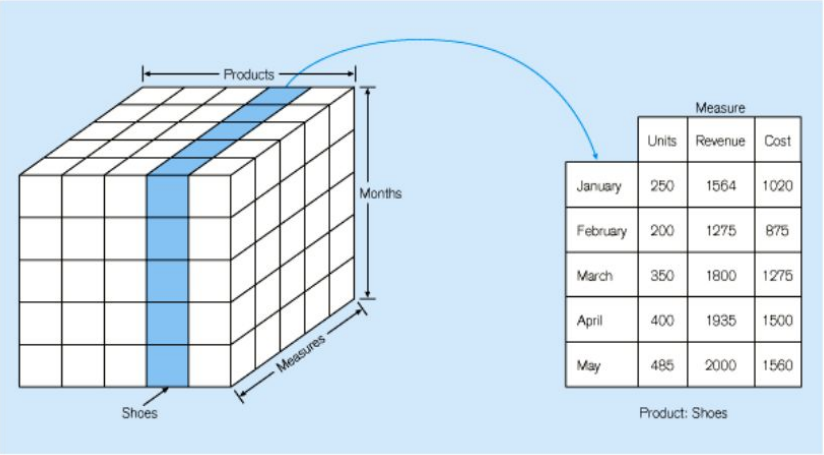
Online Analytical Processing (OLAP) is the use of a set of graphics tools that provides users with multidimensional views of their data and allows them to analyze the data using simple windowing techniques.

- A data warehouse is based on multidimensional data model, which view data in the form of data cube.
- A data cube allows data to be modeled and viewed in multiple dimensions.
  - Dimension tables
  - Fact table

MOLAP operations / A BI eszközök interaktív adatkockájának 3 fő művelete:

- **Slice and dice**: project and select
- **Drill down**: reverse of roll up, from higher level summary to lower level summary or detailed data, or introducing new dimensions.
- **Roll up**: summarize data by climbing up hierarchy or by dimension reduction

Figure 11-22: Slicing a data cube



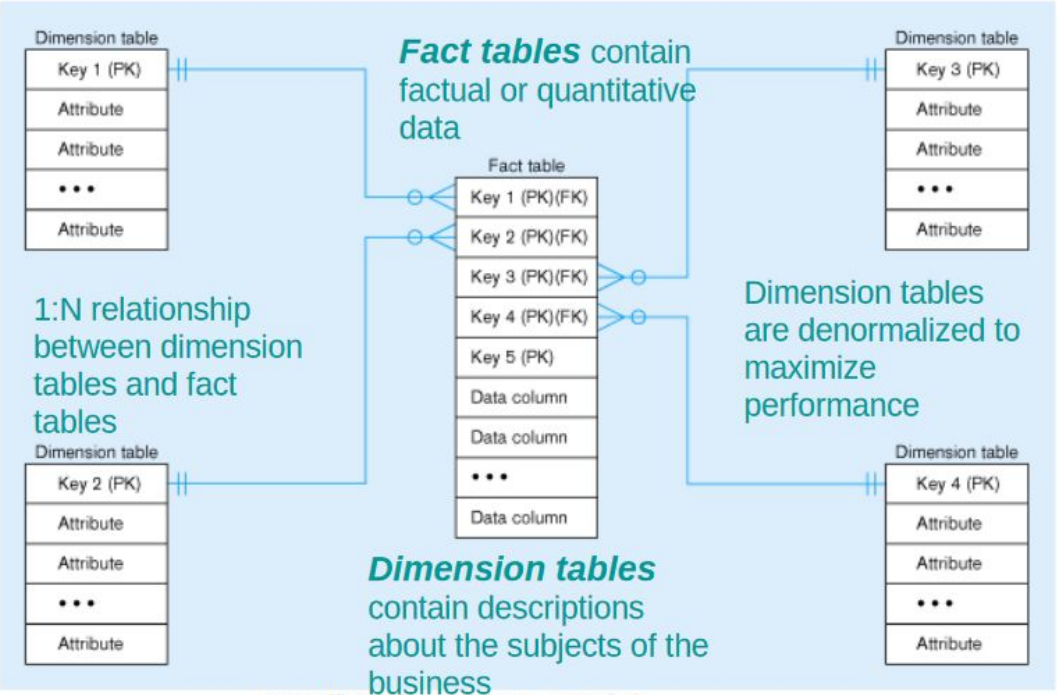
The Star Schema

Simple database design in which dimensional (describing how data are commonly aggregated) are separated from fact or event data.

Two type of tables:

- fact tables
- dimension table

Az adattárházak szokásos sémamegoldása az ún. **csillagséma**, mely **fact** táblákból és **dimenzió** táblákból áll. Ez az adatbázis séma jól támogatja az ad-hoc, interaktív lekérdezéseket. Ezt a sémamegoldást ill. ezeket a lekérdezéseket támogatják jól a **bitmap-indexek**.



Excellent for ad-hoc queries,
but bad for online transaction processing

Role of Metadata (data catalog)

- Identify subjects Identify dimensions and facts
- Indicate how data is derived from enterprise data warehouses, including derivation rules
- Indicate how data is derived from operational data store, including derivation rules
- Identify available reports and predefined queries
- Identify data analysis techniques
- Identify responsible people

Data warehouse queries

- Large number of records
- Ad-hoc queries, multiple dimensions
- Aggregations
- Read operations -

Refresh modes, query rewrite integrity:
Refresh types: Complete, Fast
Accuracy of Query Rewrite (Oracle)

- Enforced
- Trusted
- Stale_tolerated

Néhány téma amiről szó van, de nem mélyen (utána lehet azért kukkantani, ha valaki szorgalmas):
Bitmap index, Star transformation, Materialized view, query rewrite

Column-oriented DBMS

- Column-oriented
    - Aggregates over many rows and few columns
    - Compression…
- Row-oriented
    - Retrieving or changing few records with many attributes

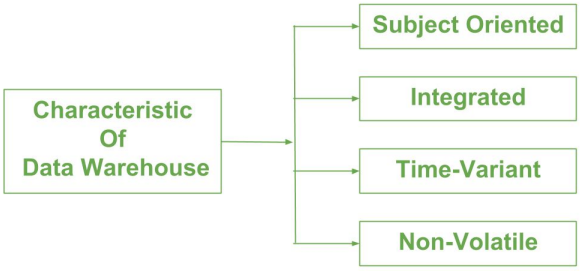
In-memory databases

- RAM: szerver 1-6 TB
- CPU magok száma, frekvencia
- SSD (logfájlok is)

SAP HANA

- High Speed Analytic Appliance
- Oszlop alapú tárolás is
    - Memóriából is gyorsabb szekvenciálisan olvasni
    - Tömörítés
- Párhuzamosítás

Data warehouse characteristics



- **Subject oriented**: the data warehousing process is proposed to handle with a specific theme which is more defined. These themes can be sales, distributions, marketing etc.
- **Integrated**: Integration means founding a shared entity to scale the all similar data from the different databases. The data also required to be resided into various data warehouse in shared and generally granted manner.
- **Time-Variant**: In this data is maintained via different intervals of time such as weekly, monthly, or annually etc.
- **Non-Volatile**: As the name defines the data resided in data warehouse is permanent. It also means that data is not erased or deleted when new data is inserted.

https://www.geeksforgeeks.org/characteristics-and-functions-of-data-warehouse/

Python - Pandas library

What is python pandas?

pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.

Advantages and disadvantages

Előnyök:

- Könnyebb fejlesztés és debuggolás
- kód olvashatóság
- úgy a python összes előnye (visualisation, machine learning)

Hátrányok

- Memória
- ACID
- cost based query optimization

Pandas segmentation and aggregation

- .loc és .iloc
- mean, group-by, sum, count etc.

Scalable data processing, NoSQL

NoSQL databases

- How much data?
    - Google 20 PB data processing daily in 2008
    - Facebook 2.5 PB user data + 15 TB daily
    - etc.
- SQL: Traditional relational DBMS (efficient, reliable, convenient, and safle multi-user storage of and access to massive amounts of persistent data)
- Recognition over past decade or so: Not every data management / analysis problem is best solved using a traditional relational DBMS (Web-based systems)
- NoSQL: (Not Only SQL) Not using traditional relational DBMS

NoSQL - Definition

Heterogeneous group of concepts, systems:

- Key-value stores
- Wide column stores
- Document stores etc.

NoSQL - Advantages

- Depend on system / category
- Higher performance
- Easy distribution of data on nodes: scalability, fault tolerance
- Flexibility: schema free data model
- Simpler administration

NoSQL - Methods

- No normalized relational data model
- Transaction management relaxed (ACID -> BASE), fewer granties
    - **Basically available**: Nodes in a distributed environment can go down, but the whole system shouldn't be affected,
    - **Soft State** (scalable): The state of the system and data change over time.
    - **Eventual consistency**: Given enough time, data will be consistent across the distributed system.

CAP Theorem

The CAP theorem or Brewer's theorem states that it is impossible for a distributed data store to simultaneously provide more than two out of the following three guarantees:

- **Consistency**: Clients should read the same data. There are many levels of consistency. Levels: strict (RDBMS), tunable (Cassandra), eventual (Amazon Dynamo)
- **Availability**: Data to be available
- **Partition tolerance**: Data to be partitioned across network segments due to network failures.

The CAP theorem implies that in the presence of a network partition, one has to choose between consistency and availability.

Google Spanner

Spanner is Google's highly available, global SQL database. It manages replicated data at great scale, both in terms of size of data and volume of transactions. It assigns globally consistent real-time timestamps to every datum written to it, and clients can do globally consistent reads across the entire database without locking.

In terms of CAP, Spanner claims to be **both consistent and highly available** despite operating over a wide area.
Partitions can happen and in fact have happened at Google, and during some partitions, Spanner chooses C and forfeits A so it is technically a CP system.

However, no system provides 100% availability, so the pragmatic question is whether or not Spanner delivers **availability that is so high that most users don't worry about its outages.**

But how Spanner achieves this high availability?

- Spanner runs on Google's private network. Google controls the entire network and thus can ensure redundancy of hardware and path, upgrades and operations.

https://cloud.google.com/blog/products/gcp/inside-cloud-spanner-and-the-cap-theorem

Key-value stores
A key-value database, or key-value store, is a data storage paradigm designed for storing, retrieving, and managing associative arrays, a data structure more commonly known today as a dictionary or hash table.

Data model: (key, value) pairs
Operations:

- Insert(key, value)
- Fetch(key)
- Update(key, value)
- Delete(key)

Implementation: efficiency, scalability, fault-tolerance

- Records distributed to nodes based on key
- Replication
- Single-record transactions "eventual consistency"

Document stores
Like key-value stores except value is document.

- XML, YAML, JSON, BSON, binary forms (PDF, docx etc.)

```
{
    "FirstName": "Bob",
    "Address": "5 Oak St.",
    "Hobby": "sailing"
}
```

```
<contact>
  <firstname>Bob</firstname>
  <lastname>Smith</lastname>
  <phone type="Cell">(123) 555-0178</phone>
  <phone type="Work">(890) 555-0133</phone>
  <address>
    <type>Home</type>
    <street1>123 Back St.</street1>
    <city>Boys</city>
    <state>AR</state>
    <zip>32225</zip>
    <country>US</country>
  </address>
</contact>
```

MongoDB

- High performance and availability
- Horizontal scalability
    - Shrading: distributing data across a cluster of machines
- Rich query language
    - Data aggregation
    - Text search
    - Geospatial queries

Wide column stores

- Key-value database
- Columns: name, format can vary from row to row
- eg. Apache Cassandra