ACID (Transaction)

Atomicity: allt körs eller inget körs.

Consistency: en transaktion ska ta databasen från ett konsekvent läge till ett annat.

Isolation: en transaktion kan inte använda andra transaktioner.

Durability: om någonting sker (så som att anslutning till databasen tappas) så bör vi se till att transaktionen körs ändå. För detta ansvarar applikationsprogrammerare.

Concurrency

Typer av problem

Lost Updates

Uppdateringar sker inte.

- Kalle har 100 SEK i ett konto (både Kalle och hans far läser)
- Kalle vill dra 30 SEK för att köpa pasta (Kalle skriver 70)
- Kalles far vill överföra 200 SEK till kontot (Kalle skriver 300)

Uncommitted Dependency (Dirty read)

Vi läser saker som inte är ordentligt uppdaterade.

- Kalle har 100 SEK i ett konto (banken läser)
- banken råkar sätta in 100 SEK (banken skriver 200, Kalle läser)
- banken tar bort 100 SEK (banken skriver 100)
- Kalle drar ut 20 SEK (Kalle skriver 180)

Inconsistent Analysis (Non-repeatable Read)

VI läser saker som bara är tillfälliga.

- T_1 och T_2 är concurrent transactions
- T_2 kalkylerar summan av konton X, Y och Z
- T_1 överför 10 SEK från kontot X till Z

- Phantom Reads

Phantom Reads

```sql
SELECT COUNT(*) AS @numberOfEmployees
    FROM Employees

UPDATE Employees
    SET salary = salary + 100000 / @numberOfEmployees
```

Mellan det att vi läser antalet anställda tills dess att vi skriver dess lön kan vi får fler eller färre anställda.

Hur vi undviker problemen

Optimistisk kontroll

Vi jobbar utan att låsa något. Vi kontrollerar manuellt innan vi "committar" en transaktion. Vi räknar inte med problem. Få användare så är det antagligen korrekt.

Pessimistisk kontroll

Vi jobbar genom att låsa en rad, tabell eller hela databasen. Våra transaktioner bör vara isolerad. Låsningar kan styra åtkomst för läsning, skrivning o.s.v. Exlusive lock (write lock), shared lock (read lock). Tidsstämpling.

MYSQL har stöd för flera olika *storage engines* så som Inno DB. Beroende på val av engine så hanteras låsning på olika vis.

Index

Primära index används för sekventiellt ordnad data efter en *unik nyckel* (PK / Primary Key). Sekundära index kan vara flera och behöver ej vara unika (FK / Foreign Keys). Har vi för många index tar det tid att räkna om index varje gång en förändring sker - desto fler desto långsammare. Om möjligt kan det vara bra att utföra många inserts (batch / bulk inserts). Man kan alltså använda en procedur som avvaktiverar indexet - lägger in all data och sedan aktiverar indexet igen.

Clustered index

- Varje rad är sorterad
- Bra för många `BETWEEN` etc.
- Maximalt ett per tabell

Non-clustered index

- Bra för höga krav på inserts
- Bra för många `WHERE`, `JOIN` etc.

Tips

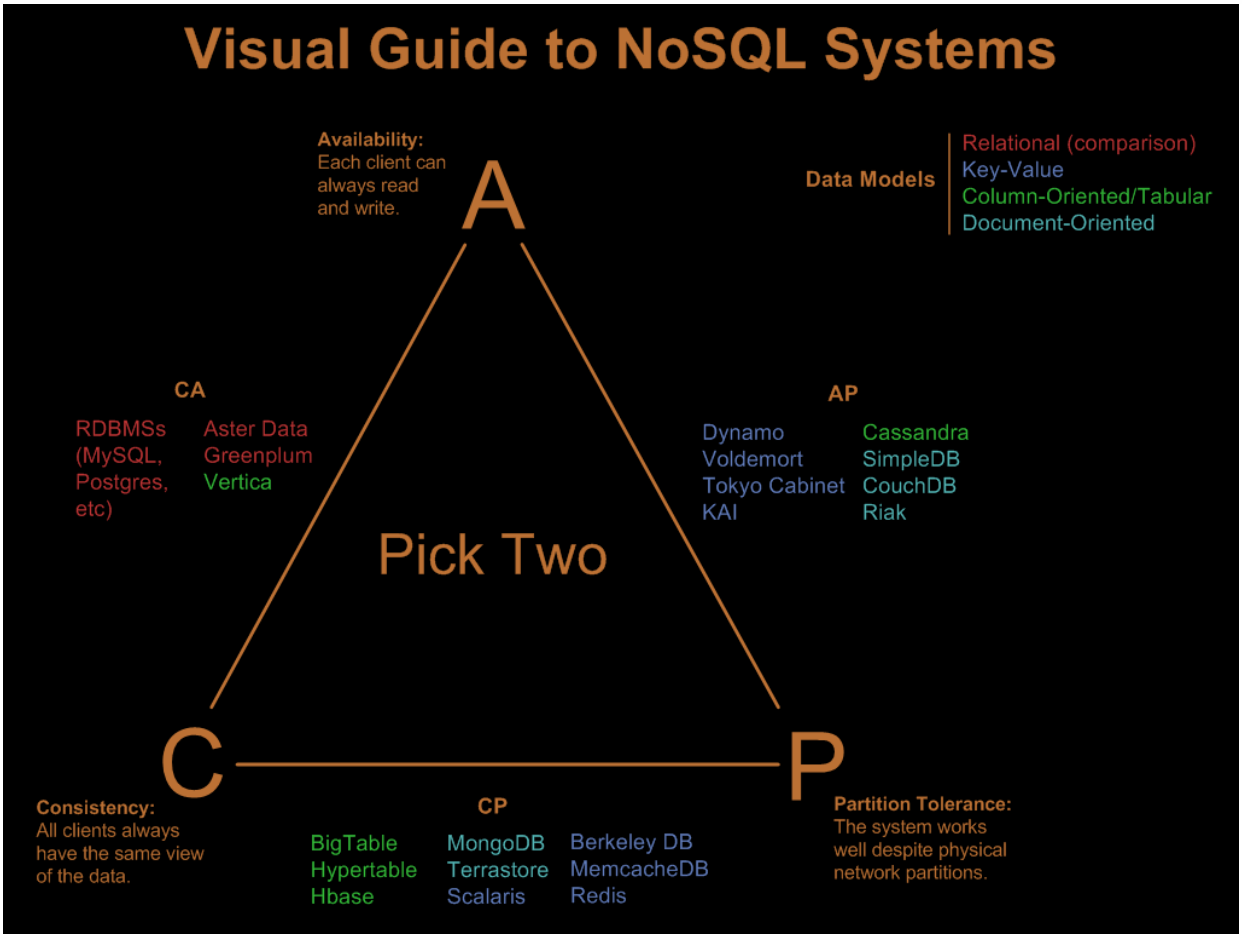
I MYSQL kan man säga att en primär nyckel är samma sak som en clustered index. Man skulle kunna säga att främmande nycklar är non-clustered index.

- Använd inte index för extremt små tabeller (de får plats i RAM-minnet)
- Lägg till sekundära index där vi använder `JOIN`, `SORT`, `ORDER BY` och andra aggregerande funktioner
- Använd `EXPLAIN`: `EXPLAIN SELECT * FROM Users`
- Undvik index för attribut som ofta uppdateras
- Undvik index för attribut som består av långa texter (`VARCHAR`)
    - aanvänd inte index där bättre lösningar finns
    - (Dock finns nya versioner av MYSQL ett s.k. FULLTEXT-index som kan fungera)
- Dokumentera val av index (`DESCRIBE` -kommandon, kommentarer, extern dokumentation)

CAP-teoremet

Det är omöjligt att tillfredställa mer än två delar i *CAP*:

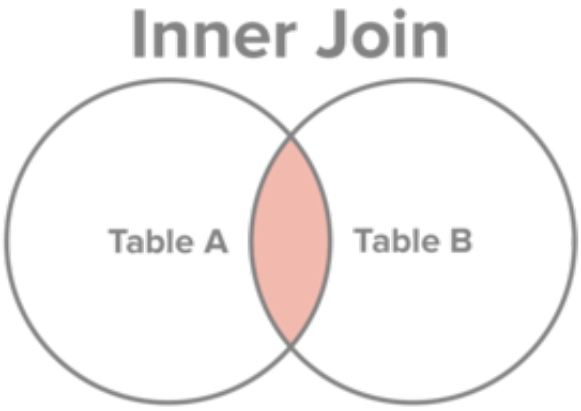
- Consistency - Varje gång vi läser får vi den senaste versionen av data eller ett felmeddelande
- Availability - Varje förfrågan får ett svar (som inte är ett felmeddelande) utan garanti för att det är den senaste versionen
- Partition tolerance - Systemet fortsätter att fungera även om en okänd mängd meddelanden tappas eller blir försenade i nätverket mellan databasens noder



(http://digbigdata.com/wp-content/uploads/2013/05/media_httpfarm5static_mevlk.png)

Join

Inner Join



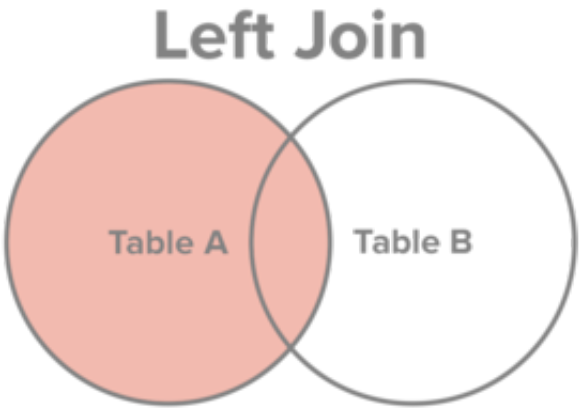
Returnerar alla rader från A och B som matchar join-villkoret.

Låt A vara en tabell över kunder och B en tabell över köp.

```
1  SELECT A.name, SUM(B.total) FROM A INNER JOIN B ON A.id = B.customer;
```

Resultatet vore kundens namn och summan av alla köp - en rad per kund och summa. Kunder som inte gjort några köp dyker inte upp i resultatet.

Left (Outer) Join



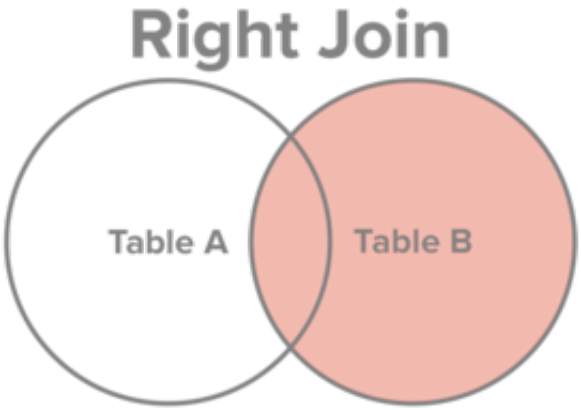
Returnera alla rader från A tillsammans med matchande rader från B om sådana finns.

Låt A vara en tabell över kunder och B en tabell över köp.

```
1  SELECT A.name, SUM(B.total) FROM A LEFT JOIN B ON A.id = B.customer;
```

Resultatet vore kundens namn och summan av alla köp - en rad per kund och summa. Kunder som inte gjort några köp är med i resultatet - men då med en summa som är `NULL`.

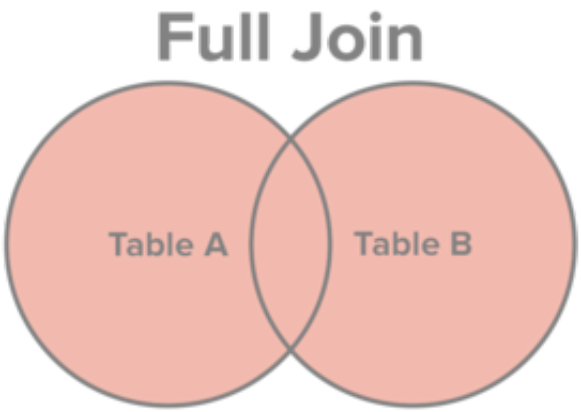
Right (Outer) Join



Returnera alla rader från B tillsammans med matchande rader från A om sådana finns.

En spegling av Left Join helt enkelt.

Full (Outer) Join



Låt A vara en tabell över kunder och B en tabell över köp.

```
SELECT * FROM A JOIN B ON A.id = B.customer;
```

Resultatet vore en rad per kund och köp. Om kunden inte gjort något köp blir värden från tabell B `NULL`. Om ett köp inte är kopplat till en kund blir värden för tabell A `NULL`.

Modellering

Normalformer

Första normalformen

En tabell är på första normalformen (1NF) om ingen attribut kan delas upp. Exempelvis skulle ett fält `name` eventuellt kunna delas upp till `surName` och `lastName`. Uppdelningen ska inte resultera i att det ena fältet beror av det andra.

Andra normalformen ⚠️

En tabell är på andra normalformen (2NF) om den är på 1NF och alla attribut beror på primärnycklen (som är sammansatt).

Tredje normalformen ⚠️

...

Boyce-Codds normalform ⚠️

Relationer får bara ske till primärnycklar.

Designstadie

1. En konceptuell nivå - leta efter generella entiteter och potentiella relationer

2. En logisk nivå - styr upp relationer och faktiska fält
3. En fysisk nivå - Implementera systemet i en DBMS

ER-modellering består av de två översta stadierna.

⚠️ Lärare har uttalat sig på olika vis / information skiljer mycket gentemot fakta från andra källor.