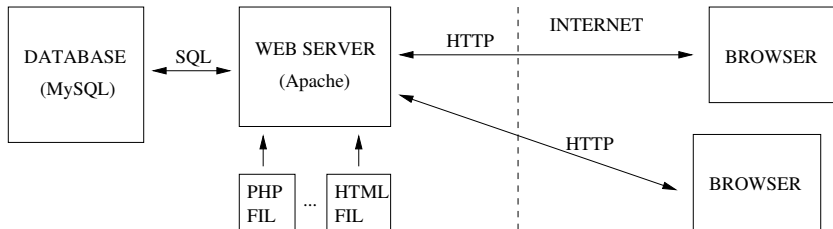


### Databaseprogrammering med SQL

- ▶ Hvad har vi lært indtil nu?
- ▶ Fire kategorier af sites
- ▶ Konstruktion af sites som er databaser
- ▶ Filsystemet som database
- ▶ Rigtige databaser — RDBMS's, the ACID test
- ▶ Databasebegreber: database, tabel, skema, felt, type, post
- ▶ Brug af MySQL-databaseserveren
- ▶ Oprettelse og udfyldning af tabeller (CREATE TABLE)
- ▶ Feltyper
- ▶ Forespørgsler (SELECT ... FROM ... WHERE ...)

# Hvad har vi lært indtil nu?

## Oversigt:



## En PHP-fil:

```
<html>
  <head><title>Hello World</title>
</head>
<body> <? echo "<b>Hello</b> ";
        echo "<i>WORLD</i>";
        ?>
</body>
</html>
```

## Indtil nu:

- ▶ Variabler, tal, strenge og arrays
- ▶ Beregninger
- ▶ if-sætninger og løkker
- ▶ Funktioner og kodegenbrug
- ▶ Indhentning af data fra brugere med forms

## 1. Traditionelle sites

- ▶ online aviser, vejrudsigter, aktiekurser
- ▶ høje omkostninger - indtjening ved reklamer, sponsorer, abonnenter

## 2. Sites som publicerer kollaborativt indsamlet information

- ▶ information opsamles elektronisk via forms
- ▶ eksempel: brugerundersøgelser

## 3. Sites som tilbyder en service via et webserver-program

- ▶ bryllups-service, GMail, CourseGrader
- ▶ indtjening ved abonnenter, fokuseret reklame

## 4. Sites som tillader brugere at tilgå flere databaser samtidigt

- ▶ bilbaser, auktionssites
- ▶ store indtjeningsmuligheder - kun udgifter til teknologien

**Bemærk:** Sites i de sidste tre kategorier er database-sites!

## Konstruktion af datamodel

- ▶ Hvilken information skal gemmes og hvordan skal den repræsenteres?
- ▶ *Dette er den svære del!*

## Udvikling af data-transaktioner

- ▶ Hvordan indsættes data i databasen?
- ▶ Hvordan udtrækkes data fra databasen?

## Udvikling af forms og PHP til implementation af transaktioner

- ▶ Brugergrensefladen er HTML-kode (forms)
- ▶ SQL (Structured Query Language) bruges til de egentlige data-transaktioner
- ▶ PHP er "limen" til at binde det hele sammen
- ▶ *Dette er den nemme del!*

## Fordele:

- ▶ Der kræves **ingen ekstra software**
- ▶ Løsning **nem at forstå** — data gemmes som tekst i filer:

mael@itu.dk

kenneth@it.edu

nh@itu.dk

gates@microsoft.com

Martin Elsman

Kenneth Riis

Niels Hallenberg

Bill Gates

## Ulemper:

1. Umiddelbare **performance**-problemer (lineær søgning)
2. Manglende **abstraktion** fra datarepræsentation
3. Problemer med håndtering af **samtidige** brugere
4. Ingen standard for **integration** med andre systemer
5. Problemer med **fejlhåndtering**

**Bemærk:** Man ender med at bygge en rigtig (R)DBMS!

## ACID-testen

1. **Atomicity.** En transaktion er enten fuldt udført eller slet ikke udført.
  - ▶ *Ved overførsel mellem bankkonti er enten begge konti opdateret eller ingen af dem opdateret.*
2. **Consistency.** En transaktion sender databasen fra en legal tilstand til en anden legal tilstand.
  - ▶ *Summen af en kundes bankkonti skal være positivt.*
3. **Isolation.** En transaktion er usynlig for andre transaktioner indtil transaktionen er komplet.
  - ▶ *Overførsel mellem bankkonti er mulig samtidig med generering af regnskabsrapport.*
4. **Durability.** Komplette transaktioner overlever fremtidige systemcrash.
  - ▶ *Når en kunde har fået bekræftet indsættelsen af et beløb på en konto vil transaktionen overleve et fremtidigt crash.*

## Bemærk:

- ▶ ACID-egenskaberne er vigtige for databasestøttede websites; specielt for sites hvor kravene til datakvaliteten er stor.
- ▶ MySQL opfylder ikke, som udgangspunkt, alle ACID kravene, som f.eks. Oracle og PostgreSQL gør det.
- ▶ Vi benytter alligevel MySQL til vores websites...
- ▶ Access-databaser implementerer **slet ikke** ACID-kravene...

En *relationsdatabase* består af en samling navngivne *tabeller*.

En *tabel* består af to dele:

▶ **Et skema** (= tabelbeskrivelse):

1. *Skemaet* bestemmer tabellens form.
2. Skemaet angiver tabellens *felder* og deres *typer* (fx: number, text).
3. Skemaet ændres sjældent.

▶ **En samling poster** (= records = tupler = tabellinier):

1. Samlingen af *poster* er tabellens *indhold* (kan variere over tid).
2. Hver post indeholder et sæt værdier for tabellens felter.
3. Rækkefølgen af posterne i en tabel er ligegyldig.

# Eksempel: Kursusdata

**Student:**

snavn	snr	adresse
Ole Olesen	L1234	Hjemmevej 7
Rikke Richardsen	L2345	Udvej 9
Søren Sørensen	J0007	I dybet 13
Ulrikka Funder	B7117	Skovvej 11

**Kursus:**

knr	knavn	lærer
W2	Databasestøttet Webpublicering	Martin Elsman
GP	Grundlæggende Programmering	Morten Rhiger

**Tilmelding:**

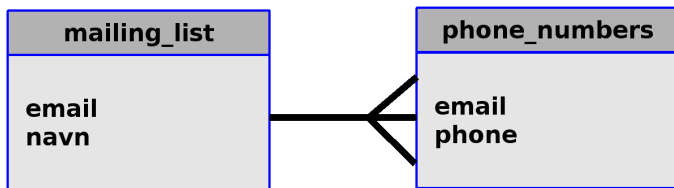
snr	knr
L1234	W2
L2345	W2
J0007	W2
J0007	GP
L2345	GP

- ▶ Der findes mange slags databaser: regneark, netværksdatabaser (DNS), object-relationelle databaser, XML-baserede databaser, ...
- ▶ Siden 1980'erne har *relationsdatabaser* fundet bred anvendelse: Banker, folkeregister, varekataloger, regnskabssystemer, medlemsregistre, studieadministration, ...

## Fordelen ved RDBMS'er: fleksible forespørgsler

- ▶ Grunddata ligger i *tabeller* der beskrives af *skemaer*.
- ▶ Udtræk, beregninger og samkøringer laves med *forespørgsler*.
- ▶ Design kan foretages **abstrakt** ud fra skemaerne.
  - ▶ Ex: Student, Kursus, Tilmelding, ... — E/R-diagrammer
- ▶ Nye forespørgsler kan udvikles når der er behov for dem (så længe datagrundlaget er der).
  - ▶ Ex: beregn gennemførelsesprocent for alle kurser, opgjort på studerendes studieretning og optagelsesår.

## E/R-diagram



- ▶ **Gaflen** i E/R-diagrammet angiver en **en-til-mange-relation**.
- ▶ Hver person, unikt identificeret ved email, kan have flere telefonnumre.

Data i `mailing_list`:

email	navn
kenneth@it.edu	Kenneth Riis
nh@it.edu	Niels Hallenberg

Data i `phone_numbers`:

email	phone
kenneth@it.edu	44 84 34 94
nh@it.edu	38 16 88 88
nh@it.edu	38 16 88 24

# Forespørgselsproget SQL

- ▶ Forespørgsler i relationsdatabaser skrives typisk i sproget SQL.
- ▶ SQL er opfundet af IBM samtidig med relationsdatabaser, ca. 1970.
- ▶ SQL benyttes i dag i alle seriøse RDBMS'er (Oracle, DB2, Postgres, MySQL, Microsoft SQL Server, ...).
- ▶ Der findes en SQL-standard (SQL-86, ..., SQL:2006), men hver implementation bidrager med egne udvidelser (f.eks. til dato-håndtering).

## Databasesystemet MySQL

- ▶ Kurset bruger databasesystemet MySQL. Det er et robust og effektivt gratis system fra [www.mysql.com](http://www.mysql.com).
- ▶ MySQL er særdeles udbredt og bruges både af private og i industrien.
- ▶ Særlige MySQL-kommandoer, udvidelser og mangler er beskrevet i:

<http://www.itu.dk/courses/DSDS/E2007/mysql.html>

# Oversigt over de vigtigste SQL-kommandoer

## Data Definition Language (design af datastruktur)

CREATE TABLE	opret tabel
CREATE UNIQUE INDEX	opret index for effektiv søgning
DROP TABLE	slet tabel
ALTER TABLE	tilføj eller modificer kolonne

## Data Manipulation Language (manipulation af data)

SELECT	foretag udtræk og samkøring
INSERT INTO ... VALUES (...)	indsæt poster i tabel
DELETE FROM	fjern poster fra tabel
UPDATE	editer poster i en tabel

- ▶ I SQL skelnes der normalt ikke mellem store og små bogstaver.
- ▶ MySQL skelner dog mellem store og små bogstaver i tabelnavne (Student er noget andet end student).
- ▶ Her skriver vi ofte SQL-ordrerne med STORT, tabelnavne med Stort begyndelsesbogstav, og feltnavne med småt.

## Eksempel på CREATE TABLE kommandoer

```
CREATE TABLE mailing_list (  
    email varchar(100) not null,  
    name  varchar(100) not null  
);
```

```
CREATE TABLE phone_numbers (  
    email varchar(100) not null,  
    phone varchar(20) not null  
);
```

## Bemærk

- ▶ not null betyder at feltet, for alle rækker, er ikke-tomt.
- ▶ varchar(100) betyder at feltet maksimalt kan være på 100 tegn.
- ▶ Det er hensigten at email-kolonnen i phone\_numbers tabellen *refererer* til et tilsvarende felt i tabellen mailing\_list: Der *bør* findes en række i mailing\_list, hvor feltet email har samme værdi som email.

# Referential Constraints

Referencebetingelsen mellem `phone_numbers.email` og `mailing_list.email` kan sikres med MySQL's InnoDB bagende:

```
CREATE TABLE mailing_list (  
    email  varchar(100) NOT NULL,  
    name   varchar(100) NOT NULL,  
    INDEX (email)  
) TYPE=InnoDB;
```

```
CREATE TABLE phone_numbers (  
    email  varchar(100) NOT NULL,  
    phone  varchar(20) NOT NULL,  
    INDEX (email),  
    FOREIGN KEY (email) REFERENCES mailing_list(email)  
) TYPE=InnoDB;
```

## Bemærk

- ▶ Fremmednøglen og den refererede nøgle skal være af samme type (streng-længder kan dog variere).
- ▶ NULL værdier tjekkes ikke for referentiel integritet.

# Oversigt over de vigtigste SQL-typer

Type	Anvendelse	Eksempel
INT	heltal	117
DOUBLE	flydende-kommatal	3.1415
VARCHAR(80)	tekst (maks 80 tegn)	'Ole Jensen'
DATE	dato	2002-03-10
TIME	klokkeslet	22:59:17
DATETIME	dato og klokkeslet	2002-03-10 22:59:17

- ▶ Der er også nogle andre typer som vi ikke skal bruge.
- ▶ Typerne bruges i CREATE TABLE kommandoer og ALTER TABLE kommandoer.

## Eksempler på INSERT kommandoer

```
INSERT INTO mailing_list (name, email)
  VALUES ('Kenneth Riis', 'kenneth@it.edu');
INSERT INTO mailing_list (name, email)
  VALUES ('Niels Hallenberg', 'nh@it.edu');
INSERT INTO phone_numbers (email, phone)
  VALUES ('kenneth@it.edu', '44 84 34 94');
INSERT INTO phone_numbers (email, phone)
  VALUES ('nh@it.edu', '35 28 23 04');
INSERT INTO phone_numbers (email, phone)
  VALUES ('nh@it.edu', '38 16 88 43');
```

## Note

- ▶ Værdier som **O'Neil** indsættes ved at *escape* tegnet ' : **O"Neil**.

## Eksempel på simpelt databaseudtræk

```
SELECT * FROM mailing_list;
```

## Eksempel på kolonnevalg

```
SELECT name FROM mailing_list;
```

## Eksempel med rækkeudvælgelse — WHERE

```
SELECT name FROM mailing_list WHERE email = 'kenneth@it.edu';
```

## Eksempel på sortering — ORDER BY

```
SELECT * FROM mailing_list ORDER BY name;
```

## Eksempel på omvendt sortering — ORDER BY ... DESC

```
SELECT * FROM mailing_list ORDER BY name DESC;
```

## Samkøring af data

- ▶ Joins bruges til at samkøre data fra forskellige tabeller.

## Eksempel på en join

```
SELECT * FROM mailing_list, phone_numbers;
```

## En bedre join

```
SELECT * FROM mailing_list, phone_numbers  
WHERE mailing_list.email = phone_numbers.email;
```

## Eller endnu bedre

```
SELECT name, mailing_list.email, phone  
FROM mailing_list, phone_numbers  
WHERE mailing_list.email = phone_numbers.email;
```

- ▶ Kommandoen DELETE bruges til at slette rækker fra en tabel:

```
DELETE FROM mailing_list WHERE email = 'nh@it.edu';
```

- ▶ **Pas på:** Husk at benytte MySQL's InnoDB "bagende" til at sikre at databasen ikke kommer i en ikke-consistent tilstand...
- ▶ Vi bør først slette fra phone\_numbers:

```
DELETE FROM phone_numbers WHERE email = 'nh@it.edu';  
DELETE FROM mailing_list WHERE email = 'nh@it.edu';
```

- ▶ Kommandoen UPDATE bruges til at opdatere kolonner i en tabel:

```
DELETE FROM phone_numbers WHERE email = 'kenneth@it.edu';  
  
UPDATE mailing_list SET email = 'kenneth@it-c.dk'  
WHERE email = 'kenneth@it.edu';
```

- ▶ Pas på WHERE-betingelserne!!!

## Opgaver

- ▶ SQL programmering i MySQL — kursusdatabase
- ▶ Ekstraopgave: modulus-11 check af cpr-numre

## Næste gang

- ▶ Forbindelse til MySQL database fra PHP...