introduction to

SCRIPTING, DATABASES, SYSTEM ARCHITECTURE

SQL II: Joins, Indices, and Group-By

Claus Brabrand

(((brabrand@itu.dk)))

Associate Professor, Ph.D.

(((Software and Systems)))

IT University of Copenhagen

ITU Online Course Evaluation

- Remember to fill in the ITU online ***course evaluation*** this week
  - (see email about this from study administration)

Agenda

- **More SQL commands**
- **Join** (samkøring af data)
- **Indices**
- **Group by**
- **Left and Right Join**

SQL: Main Commands

- **Data definition:**
  - CREATE TABLE // creates a new table
  - DROP TABLE // deletes a table
- **Data manipulation:**
  - INSERT INTO .. VALUES (..) // insert record(s)
  - SELECT .. FROM .. WHERE // retrieves information
  - DELETE FROM .. WHERE // delete record(s)
  - UPDATE .. SET .. WHERE // changes record(s)

SHOW TABLES & DESCRIBE

- Show tables in database:

```
SHOW TABLES ;
```

Tables_in_my_database
mailing_list
phone_numbers
students

- Show information about a particular table:

```
DESCRIBE students ;
```

Field	Type	Null	Key	Default	Extra
name	varchar(80)	NO		NULL	
age	int(11)	NO		NULL	

- Quit:

```
QUIT ;
```

LIMIT and ORDER-BY

- We can limit the number of records displayed:

- Retrieve a hundred recipes:

```
SELECT * FROM recipes LIMIT 100 ;
```

- Retrieve the youngest student:

```
SELECT * FROM students ORDER BY age LIMIT 1 ;
```

- Retrieve three (alphabetically) last courses:

```
SELECT * FROM courses ORDER BY name DESC LIMIT 3 ;
```

COUNT

- Given the table:

```
SELECT * FROM students ;
```

stud_id	name	age
1234	Anna	20
5678	Brian	25
9999	Claire	23

- **COUNT** (will count number of records):

```
SELECT COUNT(*) FROM students ;
```

COUNT(*)
3

- **COUNT AS** (gives the count a name):

```
SELECT COUNT(*) AS num_courses FROM students ;
```

num_courses
3

MIN, MAX, SUM, AVG

stud_id	name	age
1234	Anna	20
5678	Brian	25
9999	Claire	23

- **MIN** (minimum):

```
SELECT MIN(age) FROM students ;
```

MIN(age)
20

- **MAX** (maximum):

```
SELECT MAX(age) AS oldest FROM students ;
```

oldest
25

- **SUM**:

```
SELECT SUM(age) FROM students ;
```

SUM(age)
68

- **AVG** (average):

```
SELECT AVG(age) AS average FROM students ;
```

average
22.6667

Agenda

- More SQL commands
- **Join** (samkøring af data)
- Indices
- Group by
- Left and Right Join

Join (samkøring af data)

- Given the tables (entities):

mailing_list:

name	email
Claus	brabrand@itu.dk
Barack	obama@hotmail.com
John	jdoe@notmail.com

phone_numbers:

email	phone
brabrand@itu.dk	7218 5076
obama@hotmail.com	212-555-0000
jdoe@notmail.com	123-456-7890

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

name	email	email	phone
Claus	brabrand@itu.dk	brabrand@itu.dk	7218 5076
Barack	obama@hotmail.com	brabrand@itu.dk	7218 5076
John	jdoe@notmail.com	brabrand@itu.dk	7218 5076
Claus	brabrand@itu.dk	obama@hotmail.com	212-555-0000
Barack	obama@hotmail.com	obama@hotmail.com	212-555-0000
John	jdoe@notmail.com	obama@hotmail.com	212-555-0000
Claus	brabrand@itu.dk	jdoe@notmail.com	123-456-7890
Barack	obama@hotmail.com	jdoe@notmail.com	123-456-7890
John	jdoe@notmail.com	jdoe@notmail.com	123-456-7890

"join" operation:

gives all combos!

Join

mailing_list:

name	email
Claus	brabrand@itu.dk
Barack	obama@hotmail.com
John	jdoe@notmail.com

phone_numbers:

email	phone
brabrand@itu.dk	7218 5076
obama@hotmail.com	212-555-0000
jdoe@notmail.com	123-456-7890

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

name	email	email	phone
Claus	brabrand@itu.dk	brabrand@itu.dk	7218 5076
Barack	obama@hotmail.com	obama@hotmail.com	212-555-0000
John	jdoe@notmail.com	jdoe@notmail.com	123-456-7890

better ***"join"*** operation:

only things that are related

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

This pattern is extremely common !

name	email	phone
Claus	brabrand@itu.dk	7218 5076
Barack	obama@hotmail.com	212-555-0000
John	jdoe@notmail.com	123-456-7890

even better ***"join"*** operation:

we get the columns we are interested in

EXERCISE: Join

- Database:

suspects:

name	dna_profile
Jack the Ripper	ACTGC
Jason	ACGTC
Amagermanden	ACTTC

crimescenes:

place	item	dna_found
Living Room	Table	ACCTC
Living Room	Lamp	AGTGC
Hallway	Chainsaw	ACCTC
Back Alley	Knife	ACTGC
Bathroom	Shampoo	ACCTC
Kitchen	Milk Carton	ACTTC

- ***Join*** to determine (possibly multiple) perpetrators?

```
SELECT name, place, item
  FROM suspects, crimescenes
    WHERE dna_profile = dna_found ;
```

name	place	item
Jack the Ripper	Back Alley	Knife
Amagermanden	Kitchen	Milk Carton

EXERCISE: Join (cont'd)

- Database:

suspects:

name	dna_profile
Jack the Ripper	ACTGC
Jason	ACGTC
Amagermanden	ACTTC

crimescene:

place	item	dna_found
Living Room	Table	ACCTC
Living Room	Lamp	AGTGC
Hallway	Chainsaw	ACCTC
Back Alley	Knife	ACTGC
Bathroom	Shampoo	ACCTC
Kitchen	Milk Carton	ACTTC

- ...now also with "dna" in result:

```
SELECT name, place, item, dna_profile AS dna
  FROM suspects , crimescene
    WHERE dna_profile = dna_found ;
```

name	place	item	dna
Jack the Ripper	Back Alley	Knife	ACTGC
Amagermanden	Kitchen	Milk Carton	ACTTC

EXERCISE: Join

students:

cpr	name	phone	address
321085-1111	Anna	212-555-7755	Somewhere 4
221188-2222	Brian	212-555-5050	Anystreet 7
010190-3333	Claire	212-555-0707	Some Other Place 15
020290-4444	Danny	212-555-9999	Anywhere 1a

customers:

cpr	total
010190-3333	1,525.24
321085-1111	195.56
040587-5555	1,020,30
221188-2222	897.20

- Find names and phone numbers of students who are "good customers" (spent more than 500 kr):

```
SELECT name, phone, total
  FROM students , customers
    WHERE ( students.cpr = customers.cpr
            AND total > 500 ) ;
```

name	phone	total
Brian	212-555-5050	897.20
Claire	212-555-0707	1,525.24

Agenda

- More SQL commands
- Join (samkøring af data)
- Indices
- Group by
- Left and Right Join

How fast is a query?

- A ***join*** of two tables can take a very long time!
  - every record in one table needs to be compared to every record in the other table

```
SELECT * FROM table1 , table2 WHERE table.id1 = table.id2 ;
```

- If both tables have 10,000 records (not uncommon!), we get 10,000 x 10,000 = 100,000,000 comparisons
  - **This can easily take 20 seconds !**
- However, if `id1` and `id2` have been ***indexed***
  - **This only takes 0.05 seconds !**
  - (Note: the bigger the table, the more important with indexing)

Types of Indices (= Index'es)

- Three types of indices:

type	needs to be unique?	is allowed to be null?
`PRIMARY KEY`	*yes*	*no*
`UNIQUE`	*yes*	*yes*
`INDEX`	*no*	*yes*

  - (Note: only one PRIMARY KEY allowed per table)

- **Example:**

```
CREATE TABLE students (
    stud_id INT PRIMARY KEY,
    name VARCHAR(50) NOT NULL,
    age INT NOT NULL,
)
```

students:

stud_id	name	age
1234	Anna	20
5678	Brian	25
9999	Claire	23

When to use Indices

Indices are best used on columns that are...:

- ...frequently used in the **WHERE** part:

```
SELECT * FROM table WHERE age > 20 ;
```

- ...frequently used in the **ORDER BY** part:

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

- ...used as part of joins:

```
SELECT * FROM table1 , table2 WHERE table.id1 = table.id2 ;
```

Agenda

- **More SQL commands**
- **Join** (samkøring af data)
- **Indices**
- **Group by**
- **Left and Right Join**

GROUP BY

- Given table:
  - CREATE TABLE
  - INSERT INTO

expenses:

expense	dept	year	amount
salary	research	2001	490000
salary	sales	2002	1500000
salary	research	2002	500000
coffee	research	2003	800
coffee	sales	2003	300
salary	sales	2003	1600000
salary	research	2003	510000

- Let's calculate total expenses ***per dept.:***

```
SELECT dept, SUM(amount) FROM expenses GROUP BY dept ;
```

"GROUP BY" causes the records to be sorted and grouped wrt. their `dept` value:

expense	dept	year	amount
salary	research	2001	490000
salary	research	2002	500000
coffee	research	2003	800
salary	research	2003	510000
salary	sales	2002	1500000
coffee	sales	2003	300
salary	sales	2003	1600000

THEN the result is produced which involves calculating the 'SUM(amount)' values:

dept	SUM(amount)
research	1500800
sales	3100300

GROUP BY

- The sum of expenses grouped by ***expense and department***:

expenses:

expense	dept	year	amount
salary	research	2001	490000
salary	sales	2002	1500000
salary	research	2002	500000
coffee	research	2003	800
coffee	sales	2003	300
salary	sales	2003	1600000
salary	research	2003	510000

```
SELECT expense, dept, SUM(amount) AS total FROM expenses
  GROUP BY expense, dept ;
```

The table is first grouped by expense:

exp.	dept	year	amount
coffee	research	2003	800
coffee	sales	2003	300
salary	research	2001	490000
salary	sales	2002	1500000
salary	research	2002	500000
salary	sales	2003	1600000
salary	research	2003	510000

THEN, the table is sub-grouped by dept.:

exp.	dept	year	amount
coffee	research	2003	800
coffee	sales	2003	300
salary	research	2001	490000
salary	research	2002	500000
salary	research	2003	510000
salary	sales	2002	1500000
salary	sales	2003	1600000

FINALLY, the result is produced:

exp.	dept	total
coffee	research	800
coffee	sales	300
salary	research	1500000
salary	sales	3100000

GROUP BY

- The sum of expenses grouped by ***expense and department 2002-03***:

expenses:

expense	dept	year	amount
salary	research	2001	490000
salary	sales	2002	1500000
salary	research	2002	500000
coffee	research	2003	800
coffee	sales	2003	300
salary	sales	2003	1600000
salary	research	2003	510000

```
SELECT expense, dept, SUM(amount) AS total FROM expenses
  WHERE 2002 <= year AND year <= 2003 GROUP BY expense, dept ;
```

The table is first grouped by expense:

exp.	dept	year	amount
coffee	research	2003	800
coffee	sales	2003	300
~~salary~~	~~research~~	~~2001~~	~~490000~~
salary	sales	2002	1500000
salary	research	2002	500000
salary	sales	2003	1600000
salary	research	2003	510000

THEN, the table is sub-grouped by dept.:

exp.	dept	year	amount
coffee	research	2003	800
coffee	sales	2003	300
~~salary~~	~~research~~	~~2001~~	~~490000~~
salary	research	2002	500000
salary	research	2003	510000
salary	sales	2002	1500000
salary	sales	2003	1600000

FINALLY, the result is produced:

exp.	dept	total
coffee	research	800
coffee	sales	300
salary	research	1010000
salary	sales	3100000

EXERCISE 1

- What is the ***total amount spent per expense***?

expenses:

expense	dept	year	amount
salary	research	2001	490000
salary	sales	2002	1500000
salary	research	2002	500000
coffee	research	2003	800
coffee	sales	2003	300
salary	sales	2003	1600000
salary	research	2003	510000

```
SELECT expense, SUM(amount) FROM expenses GROUP BY expense ;
```

The table is grouped by expense:

exp.	dept	year	amount
coffee	research	2003	800
coffee	sales	2003	300
salary	research	2001	490000
salary	sales	2002	1500000
salary	research	2002	500000
salary	sales	2003	1600000
salary	research	2003	510000

THEN the result is produced which involves calculating the 'SUM(amount)' values:

expense	SUM(amount)
coffee	1100
salary	4600000

EXERCISE 2

- What is the ***average amount per expense***?

expenses:

expense	dept	year	amount
salary	research	2001	490000
salary	sales	2002	1500000
salary	research	2002	500000
coffee	research	2003	800
coffee	sales	2003	300
salary	sales	2003	1600000
salary	research	2003	510000

```
SELECT expense, AVG(amount) FROM expenses GROUP BY expense ;
```

The table is grouped by expense:

exp.	dept	year	amount
coffee	research	2003	800
coffee	sales	2003	300
salary	research	2001	490000
salary	sales	2002	1500000
salary	research	2002	500000
salary	sales	2003	1600000
salary	research	2003	510000

THEN the result is produced which involves calculating the 'AVG(amount)' values:

expense	AVG(amount)
coffeee	550
salary	920000

EXERCISE 3

- What are the ***total expenses each year***?

expenses:

expense	dept	year	amount
salary	research	2001	490000
salary	sales	2002	1500000
salary	research	2002	500000
coffee	research	2003	800
coffee	sales	2003	300
salary	sales	2003	1600000
salary	research	2003	510000

```
SELECT year, SUM(amount) FROM expenses GROUP BY year ;
```

The table is grouped by year:

expense	dept	year	amount
salary	research	2001	490000
salary	sales	2002	1500000
salary	research	2002	500000
coffee	research	2003	800
coffee	sales	2003	300
salary	sales	2003	1600000
salary	research	2003	510000

THEN the result is produced which involves calculating the 'SUM(amount)' values:

year	SUM(amount)
2001	490000
2002	2000000
2003	2111100

EXERCISE 4

- What are the ***expenses per year per department***?

expenses:

expense	dept	year	amount
salary	research	2001	490000
salary	sales	2002	1500000
salary	research	2002	500000
coffee	research	2003	800
coffee	sales	2003	300
salary	sales	2003	1600000
salary	research	2003	510000

```
SELECT year, dept, SUM(amount) AS total FROM expenses
  GROUP BY year, dept ;
```

The table is first grouped by year:

expense	dept	year	amount
salary	research	2001	490000
salary	sales	2002	1500000
salary	research	2002	500000
coffee	research	2003	800
coffee	sales	2003	300
salary	sales	2003	1600000
salary	research	2003	510000

THEN, sub-grouped by department

FINALLY, result...:

year	dept	total
2001	research	490000
2002	sales	1500000
2002	research	500000
2003	research	510800
2003	sales	1600300

EXERCISE 5

- What are the ***expenses each year in 2002-03***?

expenses:

expense	dept	year	amount
salary	research	2001	490000
salary	sales	2002	1500000
salary	research	2002	500000
coffee	research	2003	800
coffee	sales	2003	300
salary	sales	2003	1600000
salary	research	2003	510000

```
SELECT year, SUM(amount) FROM expenses
  WHERE 2002 <= year AND year <= 2003 GROUP BY year ;
```

The table is grouped by year:

expense	dept	year	amount
~~salary~~	~~research~~	~~2001~~	~~490000~~
salary	sales	2002	1500000
salary	research	2002	500000
coffee	research	2003	800
coffee	sales	2003	300
salary	sales	2003	1600000
salary	research	2003	510000

Result...:

year	SUM(amount)
2002	2000000
2003	2111100

EXERCISE 6

- What is the ***highest expenditure for each dept?***

expenses:

expense	dept	year	amount
salary	research	2001	490000
salary	sales	2002	1500000
salary	research	2002	500000
coffee	research	2003	800
coffee	sales	2003	300
salary	sales	2003	1600000
salary	research	2003	510000

```
SELECT dept, MAX(amount) FROM expenses
  GROUP BY dept ;
```

The table is grouped by dept:

expense	dept	year	amount
salary	research	2001	490000
salary	research	2002	500000
coffee	research	2003	800
salary	research	2003	510000
salary	sales	2002	1500000
coffee	sales	2003	300
salary	sales	2003	1600000

Result...:

dept	MAX(amount)
research	510000
sales	1600000

EXERCISE 7

- What is the ***highest exp. for each dept per year***?

expenses:

expense	dept	year	amount
salary	research	2001	490000
salary	sales	2002	1500000
salary	research	2002	500000
coffee	research	2003	800
coffee	sales	2003	300
salary	sales	2003	1600000
salary	research	2003	510000

```
SELECT dept, year, MAX(amount) FROM expenses
  GROUP BY dept, year ;
```

The table is first grouped by dept:

expense	dept	year	amount
salary	research	2001	490000
salary	research	2002	500000
coffee	research	2003	800
salary	research	2003	510000
salary	sales	2002	1500000
coffee	sales	2003	300
salary	sales	2003	1600000

THEN, sub-grouped by year

Result...:

dept	year	MAX(amount)
research	2001	490000
research	2002	500000
research	2003	510000
sales	2002	1500000
sales	2003	1600000

HAVING

- What is the ***highest exp. for each dept per year***?

expenses:

expense	dept	year	amount
salary	research	2001	490000
salary	sales	2002	1500000
salary	research	2002	500000
coffee	research	2003	800
coffee	sales	2003	300
salary	sales	2003	1600000
salary	research	2003	510000

```
SELECT dept, year, MAX(amount) FROM expenses
  GROUP BY dept, year HAVING MAX(amount) > 500000 ;
```

The table is first grouped by dept:

expense	dept	year	amount
salary	research	2001	490000
salary	research	2002	500000
coffee	research	2003	800
salary	research	2003	510000
salary	sales	2002	1500000
coffee	sales	2003	300
salary	sales	2003	1600000

THEN, sub-grouped by year

Result...:

dept	year	MAX(amount)
~~research~~	~~2001~~	~~490000~~
~~research~~	~~2002~~	~~500000~~
research	2003	510000
sales	2002	1500000
sales	2003	1600000

Abbreviations (aliases)

- The query...:

```
SELECT students.id, students.name
  FROM students, courses, enrollment
    WHERE students.id = enrollment.id
      AND enrollment.id = courses.id ;
```

- ...can be abbeviated to (via aliases)...:

```
SELECT s.id, s.name
  FROM students AS s, courses AS c, enrollment AS e
    WHERE s.id = e.id
      AND e.id = c.id ;
```

Agenda

- **More SQL commands**
- **Join** (samkøring af data)
- **Indices**
- **Group by**
- **Left and Right Join**

Recall "Join"

- Database:

suspects:

name	dna_profile
Jack the Ripper	ACTGC
Jason	ACGTC
Amagermanden	ACTTC

crimescenes:

place	item	dna_found
Living Room	Table	ACCTC
Living Room	Lamp	AGTGC
Hallway	Chainsaw	ACCTC
Back Alley	Knife	ACTGC
Bathroom	Shampoo	ACCTC
Kitchen	Milk Carton	ACTTC

- ***Join*** to determine (possibly multiple) perpetrators?

```
SELECT name, place, item
  FROM suspects, crimescenes
    WHERE dna_profile = dna_found ;
```

name	place	item
Jack the Ripper	Back Alley	Knife
Amagermanden	Kitchen	Milk Carton

One problem with joins

- Database:

suspects:

name	dna_profile
Jack the Ripper	ACTGC
Jason	ACGTC
Amagermanden	ACTTC

crimescenes:

place	item	dna_found
Living Room	Table	ACCTC
Living Room	Lamp	AGTGC
Hallway	Chainsaw	ACCTC
Back Alley	Knife	ACTGC
Bathroom	Shampoo	ACCTC
Kitchen	Milk Carton	ACTTC

- Suppose we instead would like to generate:

That is, always generate a record from the right table (not just when we happen to have a match)

```
SELECT place, item, name
FROM
  suspects RIGHT JOIN crimescenes
  ON dna_profile = dna_found ;
```

place	item	name (if match)
Living Room	Table	NULL
Living Room	Lamp	NULL
Hallway	Chainsaw	NULL
Back Alley	Knife	Jack the Ripper
Bathroom	Shampoo	NULL
Kitchen	Milk Carton	Amagermanden

LEFT JOIN vs. RIGHT JOIN

- Database:

suspects:

name	dna_profile
Jack the Ripper	ACTGC
Jason	ACGTC
Amagermanden	ACTTC

crimescenes:

place	item	dna_found
Living Room	Table	ACCTC
Living Room	Lamp	AGTGC
Hallway	Chainsaw	ACCTC
Back Alley	Knife	ACTGC
Bathroom	Shampoo	ACCTC
Kitchen	Milk Carton	ACTTC

- Same thing for LEFT JOIN (just swap arguments):

```
SELECT place, item, name FROM
  crimescenes LEFT JOIN suspects
  ON suspects.dna_profile =
     crimescene.dna_found ;
```

=

```
SELECT place, item, name FROM
  suspects RIGHT JOIN crimescenes
  ON suspects.dna_profile =
     crimescene.dna_found ;
```

place	item	name (if match)
Living Room	Table	NULL
Living Room	Lamp	NULL
Hallway	Chainsaw	NULL
Back Alley	Knife	Jack the Ripper
Bathroom	Shampoo	NULL
Kitchen	Milk Carton	Amagermanden

Join (normal, left, right)

a:

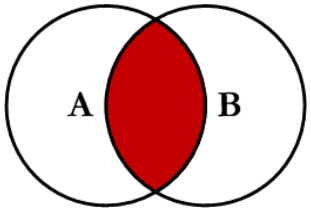
name	id
Anna	1
Brian	2
Claire	3

b:

id	course
1	DSDS
2	GSD
4	IWJX

- Normal join:

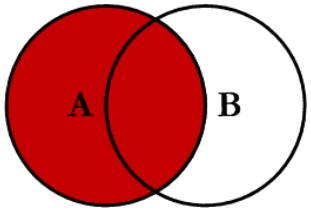
```
SELECT name, a.id, course
  FROM a , b WHERE a.id = b.id ;
```



name	id	course
Anna	1	DSDS
Brian	2	GSD

- Left join:

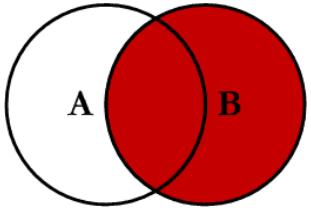
```
SELECT name, a.id, course
  FROM a LEFT JOIN b ON a.id = b.id ;
```



name	id	course
Anna	1	DSDS
Brian	2	GSD
Claire	3	NULL

- Right join:

```
SELECT name, b.id, course
  FROM a RIGHT JOIN b ON a.id = b.id ;
```



name	id	course
Anna	1	DSDS
Brian	2	GSD
NULL	4	IWJX

Assignment 7

- **Exercise 7.1:**
  - Create and fill in a course database
- **Exercise 7.2:**
  - Query the database using increasingly complex queries (7.2.1 + 7.2.2 + 7.2.3 + 7.2.4)
- **Exercise 7.3:**
  - Answer five questions on SQL

Any Questions?

(Have a nice weekend)