Complex SQL Queries

Databases

Ira Assent

ira@cs.au.dk

Data-Intensive Systems group, Department of Computer Science, Aarhus University, DK

Intended learning outcomes

- Be able to
  - Write more complex SQL queries
  - Create subqueries and joins
  - Modify the schema using SQL

ira@cs.au.dk

Recap: SQL DML

- DML: query database in SQL
- The basic form of an SQL query

`SELECT` *desired attributes*

`FROM` *one or more tables*

`WHERE` *condition about the involved rows*

https://dev.mysql.com/doc/refman/8.0/en/select.html

- INSERT INTO *table* VALUES (*list of values*);
- UPDATE *table* SET *attribute assignments* WHERE *condition*;
- DELETE FROM *table* WHERE *condition*;

https://dev.mysql.com/doc/refman/8.0/en/insert.html
https://dev.mysql.com/doc/refman/8.0/en/update.html
https://dev.mysql.com/doc/refman/8.0/en/delete.html

ira@cs.au.dk

NULL Values revisited

- An attribute value may be NULL
  - it is unknown
  - no value exists
  - it is unknown or does not exist

animal	color	zoo
lion	yellow	Copenhagen
crocodile	green	London
Tyrannosaurus Rex	NULL	NULL
polar bear	white	Berlin

- NULL values are treated specially

ira@cs.au.dk

3-Valued Logic

animal	color	zoo
lion	yellow	Copenhagen
Tyrannosaurus Rex	NULL	NULL

- Recall NULL means unknown, nonexistent, unknown or nonexistent
- Arithmetic operations on NULL yield NULL
  - 3 * NULL=NULL
  - 3 times unknown or nonexistent still unknown or nonexistent
- Any comparison with NULL yields *unknown*
  - *NULL > 3? Unknown*
- This gives 3 truth values: *true*, *false*, *unknown*
- Boolean connectives are defined appropriately

AND	**tt**	**ff**	**u**
tt	tt	ff	u
ff	ff	ff	ff
u	u	ff	u

OR	tt	ff	u
tt	tt	tt	tt
ff	tt	ff	u
u	tt	u	u

NOT	
tt	ff
ff	tt
u	u

- The WHERE clause accepts if the result is *true*

ira@cs.au.dk

What is the result?

```
SELECT userid FROM People
  WHERE office='Ny-357' OR office<>'Ny-357';
```

userid	name	group	office
ira	Ira Assent	vip	Ny-357
aas	Annika Schmidt	phd	NULL
jan	Jan Christensen	tap	Ho-017

People

1. ira, aas, jan
2. ira, jan
3. ira
4. jan
5. aas

ira@cs.au.dk

Testing for NULL

People

userid	name	group	office
ira	Ira Assent	vip	Ny-357
aas	Annika Schmidt	phd	NULL
jan	Jan Christensen	tap	Ho-017

```
SELECT userid
FROM People
WHERE office IS NULL;
```

userid
aas

Result only contains tuples that evalute to `TRUE`!

https://dev.mysql.com/doc/refman/8.0/en/working-with-null.html

ira@cs.au.dk

Textbook example

- SQL allows queries that check whether an attribute value is `NULL`
  - `IS NULL` or `IS NOT NULL`
- Retrieve the names of all employees who do not have supervisors

```
SELECT    Fname, Lname
FROM      EMPLOYEE
WHERE     Super_ssn IS NULL;
```

~~SELECT Fname, Lname~~
~~From EMPLOYEE~~
~~WHERE Super_ssn = NULL;~~

ira@cs.au.dk

Aggregation

- The SELECT clause may involve aggregate functions
  - `SUM, AVG, COUNT, MIN, MAX`
  - `NULL`s are ignored in these computations, except that `count(*)` counts all rows
- What is the average capacity of a room?

```
SELECT AVG(capacity) AS average
  FROM Rooms;
```

Rooms

room	capacity
Ny-357	6
Ada-333	26
StoreAud	286

average
106

- How many kinds of equipment do we have?

```
SELECT COUNT(DISTINCT type) as number
FROM Equipment;
```

Equipment

room	type
StoreAud	projector
StoreAud	whiteboard
Ho-017	mini-fridge
Ho-017	whiteboard

number
3

https://dev.mysql.com/doc/refman/8.0/en/aggregate-functions.html

ira@cs.au.dk

Nested Queries

- **Nested queries / subqueries**
  - Any query in parentheses can be used in FROM or WHERE clauses
    - Complete select-from-where blocks
  - **Outer query / inner query (subquery)**
    - Outer query is the first SELECT… block, the inner query is the one nested in the FROM or WHERE clause
- Example nested query: Who shares an office with Annika?

```
SELECT name
  FROM People
  WHERE office = (SELECT office
                  FROM People
                  WHERE userid='aas');
```

- A query may be used as a value if it returns only one row and one column (**scalar**)
  - Here: userid is primary key and SELECT contains a single attribute
  - Otherwise, a run-time error occurs

https://dev.mysql.com/doc/refman/8.0/en/subqueries.html

ira@cs.au.dk

Wait: but why?

- Why subqueries?
  - Very powerful: express complex conditions
    - Can use result of any query as input to another
    - **Closure** of SQL queries: relations in – relations out
  - Also use it to construct complex queries
    - Convenient
    - Start from parts of the query that you can construct, then add additional constraints
      - Caveat: may not lead to most elegant / efficient query
        - Optimization can fix some of that (coming up later), but not all…

ira@cs.au.dk

IN

- Comparison operator `IN`
  - Compares value *v* with a set (or multiset) of values *V*
  - Evaluates to `TRUE` if *v* is one of the elements in *V*
- Used to determine whether subquery results contain an element of interest to outer query

```
SELECT    DISTINCT Pnumber
FROM      PROJECT
WHERE     Pnumber IN
          ( SELECT      Pnumber
            FROM        PROJECT, DEPARTMENT, EMPLOYEE
            WHERE       Dnum=Dnumber AND
                        Mgr_ssn=Ssn AND Lname='Smith' )
          OR
          Pnumber IN
          ( SELECT      Pno
            FROM        WORKS_ON, EMPLOYEE
            WHERE       Essn=Ssn AND Lname='Smith' );
```

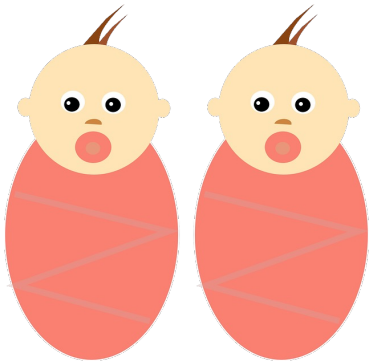
https://dev.mysql.com/doc/refman/8.0/en/comparison-operators.html

ira@cs.au.dk

IN operator for tuples

- Use tuples of values in comparisons
  - Place them in parentheses
  - Number of elements and schema needs to match
    - Essentially the subquery is a table and is used like one
    - Corresponds to multiset of values

```
SELECT    DISTINCT Essn
FROM      WORKS_ON
WHERE     (Pno, Hours) IN ( SELECT    Pno, Hours
                            FROM      WORKS_ON
                            WHERE     Essn='123456789' );
```



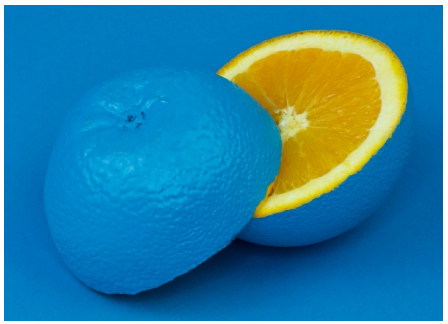
ira@cs.au.dk

ANY and ALL

- Allow comparisons against
  - *any* row in a subquery (`ANY` /`SOME`)
    - If = used as comparison operator: returns `TRUE` if the value *v* is equal to some value in the set *V* and is hence equivalent to `IN`
    - Other operators that can be combined with `ANY` (or `SOME`): >, >=, <, <=, and <>
  - *all* rows in a subquery (`ALL`)

- Which are the latest meetings that are planned?

```
SELECT topic
  FROM Meetings
  WHERE date >= ALL(
    SELECT date FROM Meetings
  );
```



https://dev.mysql.com/doc/refman/8.0/en/any-in-some-subqueries.html

ira@cs.au.dk

Types of Nested Queries

- **Correlated** nested query
  - Condition in WHERE clause of the subquery references some attribute of a relation declared in outer query
  - Evaluated once for each tuple in the outer query

```
SELECT E.Lname FROM Employee E WHERE salary > (SELECT
AVG(salary) FROM Employee WHERE Dno = E.Dno));
```

  - In order to execute the subquery you need to know the *department number of the outer query* so that you can compute the respective average *for that particular* department
  - Compare with

```
SELECT Lname FROM EMPLOYEE WHERE Salary > (SELECT
AVG(Salary) FROM EMPLOYEE);
```

    - You can run the subquery independently of the outer query, the average is the same for any tuple of the outer query – NOT correlated

https://dev.mysql.com/doc/refman/8.0/en/correlated-subqueries.html

ira@cs.au.dk

Correlated Subqueries example

- Condition in WHERE clause of the subquery references some attribute of a relation declared in outer query
- Evaluated once for each tuple in outer query

Meetings

meetid	date	owner	topic
34716	2023-08-28	ira	DB
34717	2024-01-22	ira	DB
42835	2023-08-18	aas	Prog

Which meetings exceed the capacity of a room?

```
SELECT meetid
  FROM Meetings
  WHERE (SELECT COUNT(DISTINCT pid) FROM Participants
         WHERE meetid=Meetings.meetid AND
               status<>'d' AND
               pid NOT IN (SELECT room
                           FROM Rooms) )
        (SELECT capacity
         FROM Rooms, Participants
         WHERE room=pid AND meetid=Meetings.meetid);
```

Participants

meetid	pid	status
34716	StoreAud	a
34716	ira	a
42835	zoffe	d

Rooms

room	capacity
Ny-357	6
Ada-333	26
StoreAud	286

ira@cs.au.dk

Correlated nested queries textbook example

- Condition in WHERE clause of the subquery references some attribute of a relation declared in outer query
- Evaluated once for each tuple in the outer query
- Which employees have a dependent with the same first name and sex as the employee?

```
SELECT    E.Fname, E.Lname
FROM      EMPLOYEE AS E
WHERE     E.Ssn IN  ( SELECT    Essn
                      FROM      DEPENDENT AS D
                      WHERE     E.Fname=D.Dependent_name
                                AND E.Sex=D.Sex );
```

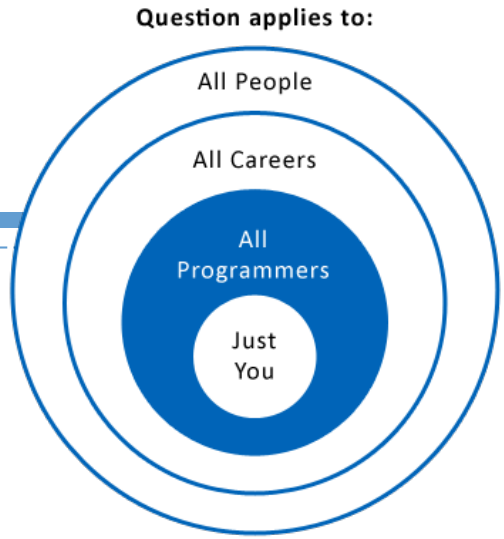
  - For each row in Employee take its Ssn value to find result of subquery for it
    - First row: Ssn 123456789, check if there is a matching dependent row where John's first name is the dependent's first name and John's sex is the dependent's sex (so a male dependent called John)
    - Next row, check if there is a male dependent Franklin
    - ….

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4

ira@cs.au.dk

Ambiguity in nested queries



- As a rule, ambiguous attribute names in outer query and subquery allowed
  - Unqualified attribute refers to innermost query
  - Similar to scope rules in programming languages
- Avoid potential errors and ambiguities by renaming
  - Create tuple variables (aliases) for tables referenced in SQL query

```
SELECT    E.Fname, E.Lname
FROM      EMPLOYEE AS E
WHERE     E.Ssn IN  ( SELECT    Essn
                      FROM    DEPENDENT AS D
                      WHERE   E.Fname=D.Dependent_name
                              AND E.Sex=D.Sex );
```

  - `E.Sex=D.Sex` need variable because both `Dependent` (innermost) and `Employee` have a `Sex` attribute,
  - `E.Fname` variable not necessary, could be just `Fname` (no `Fname` in `Dependent`)

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship

ira@cs.au.dk

Equivalence of queries

- Nested query with select-from-where blocks using = or `IN` as comparison: has equivalent query (i.e. query with identical result) with single select-from-where block
  - The comparison is "translated" to join conditions
  - Simpler, often more efficient

```
SELECT    E.Fname, E.Lname
FROM      EMPLOYEE AS E
WHERE     E.Ssn IN  ( SELECT    Essn
                      FROM    DEPENDENT AS D
                      WHERE   E.Fname=D.Dependent_name
                              AND E.Sex=D.Sex );
```

```
SELECT    E.Fname, E.Lname
FROM      EMPLOYEE AS E, DEPENDENT AS D
WHERE     E.Ssn=D.Essn
          AND E.Fname=D.Dependent_name
          AND E.Sex=D.Sex;
```

ira@cs.au.dk

The EXISTS and UNIQUE Functions in SQL

- `EXISTS` function
  - Check whether the result of a correlated nested query is empty or not
- `EXISTS` and `NOT EXISTS`
  - Typically used in conjunction with a correlated nested query
- SQL function `UNIQUE(Q)`
  - Returns `TRUE` if there are no duplicate tuples in the result of query Q

- Who is alone in an office?

```
SELECT name
  FROM People p1
  WHERE NOT EXISTS (
     SELECT *
     FROM People
     WHERE office = p1.office AND
           userid <> p1.userid
  );
```

https://dev.mysql.com/doc/refman/8.0/en/exists-and-not-exists-subqueries.html

ira@cs.au.dk

Deletion semantics

Suppose that only Annika and Chris share an office

Suppose in the loop we come to Chris first

Subquery is nonempty, because of Annika, so delete Chris

Now, moving to Annika, do we delete that tuple too?

```
DELETE FROM People p
    WHERE EXISTS (
        SELECT * FROM People
         WHERE office =
                p.office
         AND userid <>
                p.userid  );
```

1. We do not delete Annika because it does not meet the condition when we get to that row.
2. We delete Annika as well because it meets the condition when we get to that row.
3. We delete Annika as well because it was originally part of the matched tuples before deletion starts.
4. We do not delete Annika because it was not originally part of the matched tuples before deletion starts.

ira@cs.au.dk

The JOIN Operator

```
SELECT * FROM Table_1 JOIN Table_2 ON condition;
```

is syntactic sugar for:

```
SELECT * FROM Table_1,Table_2 WHERE condition;
```

```
SELECT * FROM Course JOIN Attends ON
  Course.courseid=Attends.courseid;
```

Powered by Dall-e 3

Permits users to specify a table resulting from a join operation in the FROM clause of a query

Often easier to read and distinguish from conditions that do not relate to a join of tables

```
SELECT   Fname, Lname, Address
FROM     (EMPLOYEE JOIN DEPARTMENT ON Dno=Dnumber)
WHERE    Dname='Research';
```

https://dev.mysql.com/doc/refman/8.0/en/join.html

ira@cs.au.dk

INNER JOINs

- **Inner join**
  - Default type of join in a joined table
  - Tuple is included in the result only if a matching tuple exists in the other relation
  - What we have considered so far, and what happens if you list join tables in FROM clause and join condition in WHERE clause
- **NATURAL JOIN** on two relations
  - No join condition specified
  - "naturally" joined on matching values of matching attributes
  - i.e., implicit **EQUIJOIN** condition (=) for each pair of attributes with same name from the two relations
    - "duplicate" columns only returned once
  - `SELECT * FROM Course NATURAL JOIN Attends;`
    - If `Course` and `Attends` both (only) share attribute `courseid`, join on matching ids
      - If more than one shared attribute, match needs to be on values of all shared attributes, e.g. if both have attribute `name`, join on match of `courseid AND name`
  - Like equijoin `SELECT * FROM Course JOIN Attends ON Course.courseid=Attends.courseid;`

ira@cs.au.dk

Dangling Rows and FULL JOIN

- Consider `T1 JOIN T2 ON condition`
  - A row in T1 or T2 that does not match a row in the other table is **dangling**
  - An ordinary JOIN (INNER JOIN) throws away dangling rows
- An OUTER JOIN preserves dangling rows by padding them with NULL values
  - A LEFT or RIGHT JOIN preserves dangling rows from one table only
  - Note: in MySQL, no outer join, only left or right join (use union of both to get outer join)
- In which offices are meetings planned?
  - All offices and their meeting information, if any

```
SELECT office, meetid
  FROM People LEFT JOIN Participants
                ON pid=office;
```

  - Only those offices with meetings

```
SELECT office, meetid
  FROM People JOIN Participants
                ON pid=office;
```

Participants

meetid	pid	status
34716	StoreAud	a
34716	ira	a
42835	zoffe	d

https://dev.mysql.com/doc/refman/8.0/en/join.html

ira@cs.au.dk

The JOIN Operator

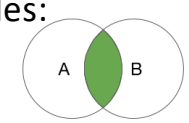
Example: equality join on "name" attributes in these two tables

id	name
1	Joe
2	Ann

TAid	name	course
111	Jane	DB
222	Joe	Prog

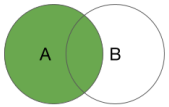
id	name	TAid	name	course
1	Joe	222	Joe	Prog

Inner equality join on "name" attributes in the two tables: Only match "Joe"



INNER JOIN

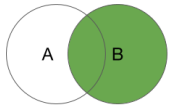
id	name	TAid	name	course
1	Joe	222	Joe	Prog
2	Ann	NULL	NULL	NULL



LEFT OUTER JOIN

Left outer join: pad with NULL values for those rows in the left table that do not have a match ("Ann")

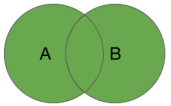
id	name	TAid	name	course
1	Joe	222	Joe	Prog
NULL	NULL	111	Jane	DB



RIGHT OUTER JOIN

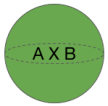
Right outer join: pad with NULL values for those rows in the right table that do not have a match ("Jane")

Full outer join: pad with NULL values for all rows in both tables that do not have a match



FULL OUTER JOIN

id	name	TAid	name	course
1	Joe	222	Joe	Prog
2	Ann	NULL	NULL	NULL
NULL	NULL	111	Jane	DB



CARTESIAN (CROSS) JOIN

Cartesian product: combine all pairs of rows from the two tables, no matching condition (so not really any equality join here…)

id	name	TAid	name	course
1	Joe	222	Joe	Prog
2	Ann	222	Joe	Prog
1	Joe	111	Jane	DB
2	Ann	111	Jane	DB

ira@cs.au.dk

IN versus Join

```
SELECT name
FROM Student , Course
WHERE studies = type;
```

name	studies
Gry	CS
Ben	DS

Student

type	code
CS	17
CS	42

Course

```
SELECT name
FROM Student
WHERE studies IN (SELECT type FROM Course);
```

1. Join outputs Gry once, IN outputs Gry twice
2. Join outputs Gry twice, IN outputs Gry once
3. Both output Gry once
4. Both output Gry twice

ira@cs.au.dk

Grouping: The GROUP-BY clause

- `SELECT-FROM-WHERE-`**`GROUP BY`**
  - Rows are grouped by a set of attributes
  - Aggregations in SELECT are done for each group
- The attributes in SELECT must be either
  - aggregates or
  - mentioned in the GROUP BY clause

- How many meetings has each person arranged?

```
SELECT owner, COUNT(meetid) as number
  FROM Meetings
  GROUP BY owner;
```

owner	number
ira	2
aas	1

https://dev.mysql.com/doc/refman/8.0/en/group-by-handling.html
ira@cs.au.dk

Grouping creates partitions

- **Partition** relation into subsets of tuples
  - Based on **grouping attribute(s)**
  - Apply function to each such group independently
- **`GROUP BY`** clause
  - Specifies grouping attributes
- If `NULL`s exist in grouping attribute
  - Separate group created for all tuples with a NULL value in grouping attribute

```
SELECT study, AVG(grade)
FROM Grades
GROUP BY study;
```

  - For each distinct value of attribute study what is the aggregate (average grade)
    - Returns e.g. average grade of CS students, of DS students, …

ira@cs.au.dk

Conditions on groups: the HAVING clause

- A HAVING clause specifies conditions for groups (otherwise, group is eliminated from result)
  - Attributes in HAVING must be aggregates or mentioned in GROUP BY (or functionally dependent, we'll get to that…)
- Which offices have more than one occupant?

```
SELECT office
  FROM People
  GROUP BY office
  HAVING COUNT(*) > 1;
```

ira@cs.au.dk

Textbook example GROUP BY and HAVING

- **`HAVING`** clause
  - Provides a condition on the summary information

Query 28. For each department that has more than five employees, retrieve the department number and the number of its employees who are making more than $40,000.

```
Q28:  SELECT   Dnumber, COUNT (*)
      FROM     DEPARTMENT, EMPLOYEE
      WHERE    Dnumber=Dno AND Salary>40000 AND
               ( SELECT       Dno
                 FROM         EMPLOYEE
                 GROUP BY Dno
                 HAVING       COUNT (*) > 5)
```


ira@cs.au.dk

Advanced Updates: Inserting a Subquery

- Invite everyone whom Frank meets with to his Beer tasting


```
INSERT INTO Participants (
      SELECT 48333 AS meetid, pid, 'u'
      FROM Meetings NATURAL JOIN Participants
      WHERE owner = 'fra'
            AND pid <> 'fra'
            AND pid NOT IN (SELECT room FROM Rooms));
```

Natural join on meetid: all participants for each meeting, add everyone who is not Frank nor a room to Participants

Meetings

meetid	date	owner	topic
34716	2023-08-28	ira	DB
48333	2025-02-11	fra	Beer tasting

Participants

meetid	pid	status
34716	StoreAud	a
48333	basement	a
48333	fra	a

Rooms

room	capacity
Ny-357	6
basement	500
StoreAud	286

ira@cs.au.dk

Which works?

1. ```
   SELECT shop FROM Sells
      GROUP BY song
      HAVING COUNT(song)<2;
   ```
2. ```
   SELECT shop, COUNT(song) FROM
   Sells   WHERE COUNT(song)<2;
   ```
3. ```
   SELECT shop, COUNT(song),
   AVG(price)    FROM Sells
      GROUP BY shop
      HAVING COUNT(song)<2;
   ```
4. ```
   SELECT shop, AVG(song) FROM Sells
      GROUP BY shop
      HAVING COUNT(song)<2;
   ```

Sells

shop	song	price
PearMusic	Flowers	1
Hotify	Flowers	2
PearMusic	Faduma	1
UTooba	People	.5
PearMusic	People	.7

ira@cs.au.dk

Schema Change Statements in SQL

- **Schema evolution commands**
  - Can be done while the database is operational
  - Does not require recompilation of the database schema

- `DROP` command
  - Used to drop named schema elements, such as tables, domains, or constraint
- Drop behavior options:
  - `CASCADE` and `RESTRICT`
    - `CASCADE` also drops constraints and other elements that reference a dropped table etc.
    - `RESTRICT` means drop only if it has no elements
- Example:
  - `DROP TABLE DEPARTMENT CASCADE;`

ira@cs.au.dk

The ALTER Command

- **Alter table actions** include:
  - Adding or dropping a column (attribute)
  - Changing a column definition
  - Adding or dropping table constraints

- Example:
  - `ALTER TABLE COMPANY.EMPLOYEE ADD COLUMN Job VARCHAR(12);`
- Change constraints specified on a table
  - Add or drop a named constraint
- To drop a column
  - Choose either `CASCADE` or `RESTRICT`

```
ALTER TABLE COMPANY.EMPLOYEE
DROP CONSTRAINT EMPSUPERFK CASCADE;
```


ira@cs.au.dk

Advanced Updates: what works?

1. UPDATE Employee SET (max(salary), min(bonus) FROM Employee WHERE Dnumber=1);
2. UPDATE Employee SET (salary, bonus) = (SELECT max(salary), min(bonus) FROM Employee WHERE Dnumber=1);
3. UPDATE Employee SET (salary, bonus FROM Employee WHERE Dnumber=1);
4. UPDATE Employee SET (salary, bonus) = (SELECT salary, bonus FROM Employee WHERE Dnumber=1);

ira@cs.au.dk

SQL is Everywhere

ira@cs.au.dk

Summary

- Intended learning outcomes
- Be able to
  - Write more complex SQL queries
  - Create subqueries and joins
  - Modify the schema using SQL

ira@cs.au.dk

Where to go from here?

- We know SQL! (just need a bit of practice…)

- Troubleshooting
  - `Use database MyDB;` // tell MySQL which database (MyDB) we are working with
  - Use ´ as escape character when defining names that are protected, e.g. `CREATE TABLE ´GROUP´`… because `GROUP BY` is a keyword

- We turn to the question "What is normal?"
  - Controlling redundancy: normal forms
  - Understanding dependencies to obtain normal forms
  - (Further) improving database schemas

ira@cs.au.dk

What was this all about?

Guidelines for your own review of today's session

- Using aggregates, functions, removing / maintaining duplicates and avoiding name clashes by…
- A subquery can be used as follows…
  - Correlated subqueries are…
- We have different types of joins that allow…
  - They differ from / are related to subqueries in that…
- Instead of single rows, we can form groups…
  - When specifying conditions on groups…
- We can change the schema as follows…

ira@cs.au.dk