SQL
Elmasri/Navathe ch 4
Padron-McCarthy/Risch ch 7/8/9
DATABASE DESIGN I - 1DL300 Fall 2012
Sobhan badiozamany
Silvia Stefanova
Department of Information Technology, Uppsala University

Note:
This is an introduction in form of a tutorial. For the course&labs more SQL knowledge is needed. Official slides from the book will be available at Studentportalen.
SQL
Elmasri/Navathe ch 4
Padron-McCarth/Risch ch 7/8/9

DATABASE DESIGN I - 1DL300 Fall 2012
Sobhan badiozamany
Silvia Stefanova
Department of Information Technology, Uppsala University

Note:
This is an introduction in form of a tutorial. For the course&labs more SQL knowledge is needed. Official slides from the book will be available at Studentportalen.
Structured Query Language

The subset of SQL that is used in creating, changing, and deleting the description (schema) of tables.

Queries

SELECT
FROM
WHERE
GROUP BY
HAVING
ORDER BY

Data Definition Language

This is an introduction in form of a tutorial. For the course & labs more SQL knowledge is needed. Official slides from the book will be available at Studentportalen.
Data Definition Language

The subset of SQL that is used in creating, changing and deleting the description (schema) of tables.

Create table

Specify a new table by giving it a name and specifying its columns and the data type (domain) of the columns.

Alter table

Modify the definition of a table by adding, dropping, or changing columns and constraints.

Drop table

Remove the definition of a table from the database schema, which includes:
- The table itself will no longer be accessible.
- Data in the table will be lost.

DROP TABLE table_name;
Specifies a new table by giving it a name and specifying its columns and the data type (domain) of the columns.

```
CREATE TABLE DEPARTMENT(
    DNAME VARCHAR(10) NOT NULL,
    DNUMBER INTEGER NOT NULL,
    MGRSSN CHAR(9),
    PRIMARY KEY (DNUMBER),
    MGRSTARTDATE CHAR(9)
);
```

Let's try it!
The concept of "Country" can be modeled as:

CREATE TABLE Country (CNAME VARCHAR(50), CNUMBER INT, PRIMARY KEY (CNUMBER));

An example:

INSERT INTO Country (CNAME, CNUMBER) VALUES ('USA', 1001);

Let's create a table for "Parts":

CREATE TABLE Parts (PNUMBER INT, PNAME VARCHAR(50), PRIMARY KEY (PNUMBER));

An example:

INSERT INTO Parts (PNUMBER, PNAME) VALUES (1001, 'Part A');

The relationships between the tables can be modeled as:

employee
- head_of: dept
- manages: employee

dept
- subdivision: store

store
- location: city

item
- quantity: debit
- deliver: supplier

supplier
- supply: parts

parts
- number: parts

record
- number: record
- sdate: account
- name: record
- price: record
The company decides to expand internationally . . .

What is the cardinality ratio of "Belongs to" relationship?

Lets create a table for the "Country" entity . . .

An example:
creating a department table.

CREATE TABLE DEPARTMENT(
    DNAME    VARCHAR(10) NOT NULL,
    DNUMBER  INTEGER NOT NULL,
    MGRSSN   CHAR(9),
    PRIMARY KEY (DNUMBER),
    MGRSTARTDATE CHAR(9)
);

name
Language
population
Let's create a table for the "Country" entity . . .

An example: creating a department table.

```
CREATE TABLE DEPARTMENT(
    DNAME    VARCHAR(10)  NOT NULL,
    DNUMBER  INTEGER      NOT NULL,
    MGRSSN   CHAR(9),
    PRIMARY KEY (DNUMBER),
    MGRSTARTDATE CHAR(9)
);
```
Alter table

Is used to change definition of tables, such as adding/removing:
- attributes
- foreign keys.
Adding a job title attribute to the employee table:

ALTER TABLE EMPLOYEE
ADD JOB_TITLE VARCHAR(12);
Adding the foreign key to dept:
ALTER TABLE dept ADD CONSTRAINT fk_dept_store
FOREIGN KEY (store) REFERENCES store (number);

Dropping the same foreign key from dept:
ALTER TABLE dept
DROP FOREIGN KEY fk_dept_store;
Alter table

Is used to change definition of tables, such as adding/removing:
- attributes
- foreign keys.

Let's try it!
The company decides to expand internationally . . .

How should we implement the "Belongs to" relationship?

To implement (N-1) relationships, we need a foreign key attribute in the entity that is on the N side, here being "city".

A side note:
A common mistake is to draw foreign keys as [oval shaped] attributes in ER. The relationship (here "Belongs to") represents the concept of foreign key.
Add a "country" attribute to city, similar to:

```sql
ALTER TABLE [table_name]
ADD [Attribute_X] [Data type];
```

What should be the data type of this "country" attribute?
The company decides to expand internationally . . .

How should we implement the "Belongs to" relationship?

A side note:
A common mistake is to draw foreign keys as [oval shaped] attributes in ER. The relationship (here "Belongs to") represents the concept of foreign key.

To implement (N-1) relationships, we need a foreign key attribute in the entity that is on the N side, here being "city".
Define a foreign key constraint that enforces the relationship, similar to:

```
ALTER TABLE [table_name]
ADD CONSTRAINT [fk_name] FOREIGN KEY ([fk_attribute])
REFERENCES [referenced table] ([pk_attribute]);
```
The company decides

How should our database be structured?

A side note:
A common mistake is to draw foreign keys as [oval shaped] attributes in ER. The relationship (here "Belongs to") represents the concept of foreign key.

Add a "country" attribute to city, similar to:

```
ALTER TABLE [table_name]
ADD
[Attribute X] [Data type];
```

What should be the data type of this "country" attribute?
Drop table

Removes the definition of table from the database schema, which means:
  • The table will no longer be accessible.
  • Data in the table would be lost.

DROP TABLE  table_name;
Data Definition Language

The subset of SQL that is used in creating, changing and deleting the description (schema) of tables.
Data Manipulation Language

The subset of SQL used in adding, modifying and removing rows in the tables.
Insert

Used in inserting rows into a table.

Inserting by specifying a row

```
<table>
<thead>
<tr>
<th>NAME</th>
<th>DEPT</th>
<th>SAL</th>
<th>BONUS</th>
<th>ENAME</th>
<th>CITY</th>
<th>STATE</th>
<th>CODE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

- Insert into `EMPLOYEES` VALUES ('Richard', 'Mark', 55250, 1000, 'Mr. Oak Forest, TX', 'TX', 75093, 75093, 31);
  - The order of attribute values should match the order of attributes in CREATE TABLE command.

```
INSERT INTO EMPLOYEE (ENAME, INAME, SDALE)
VALUES ('Richard', 'Mark', 55250);
```
- When inserting into a table, the order of values should match the order specified within the command.
- NULL will be assigned to unspecified attributes.

Inserting results of a query

Suppose we want to copy data from one table to another...

```
INSERT INTO DEPTS_INFO
(DEPT_NAME, NO_OF_EMP; TOTAL_SAL)
SELECT
ENAME, COUNT(*) SUM(SALARY)
FROM
DEPARTMENT, EMPLOYEE
WHERE
ENUMBER = DNO
GROUP BY
ENAME;
```

First, write a query that produces the results that you are going to insert.

Then insert the result into destination table.

Prezi
**Inserting by specifying a row**

<table>
<thead>
<tr>
<th>EMPLOYEE</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>FNAME</td>
<td>MINIT</td>
<td>LNAME</td>
<td>SSN</td>
<td>BDATE</td>
<td>ADDRESS</td>
<td>SEX</td>
<td>SALARY</td>
<td>SUPERSSN</td>
<td>DNO</td>
</tr>
</tbody>
</table>

```sql
INSERT INTO EMPLOYEE VALUES
('Richard','K','Marini', 653298653, '30-DEC-52','98 Oak Forest,Katy,TX', 'M', 37000,987654321, 4);
```
- The order of attribute values should match the order of attributes in create table command.

```sql
INSERT INTO EMPLOYEE (FNAME, LNAME, SSN)
VALUES ('Richard', 'Marini', 653298653);
```
- Here the order of values should match the order specified within the command.
- NULL will be assigned to unspecified attributes.

**What happens if Null value is not allowed for an unspecified attribute?**

*Let's try it!*

*The Diagram shows a flowchart with steps for inserting data into a table.*
Inserting by specifying a row

**EMPLOYEE**

<table>
<thead>
<tr>
<th>FNAME</th>
<th>MINIT</th>
<th>LNAME</th>
<th>SSN</th>
<th>BDATE</th>
<th>ADDRESS</th>
<th>SEX</th>
<th>SALARY</th>
<th>SUPERSSN</th>
<th>DNO</th>
</tr>
</thead>
</table>

**INSERT INTO EMPLOYEE VALUES**

('Richard','K','Marini', 653298653, '30-DEC-52','98 Oak Forest,Katy,TX', 'M', 37000,987654321, 4);

- The order of attribute values should match the order of attributes in create table command.

**INSERT INTO EMPLOYEE (FNAME, LNAME, SSN) VALUES** (‘Richard', 'Marini', 653298653);

- Here the order of values should match the order specified within the command.
- NULL will be assigned to unspecified attributes.

What happens if Null value is not allowed for an unspecified attribute?

*Prezi-audio*
Now let's insert two countries into our newly created "country" table:
  • USA with population of 314,000,000, an English speaking country
  • Sweden with population of 9,000,000 a Swedish speaking country

Then insert two Swedish cities in the city table:
  • Uppsala
  • Stockholm

Recall that the city table has three attributes:
  • Name
  • State
  • Country

An example insert command:

```
INSERT INTO EMPLOYEE (FNAME, LNAME, SSN) VALUES ('Richard', 'Marini', 653298653);
```
Now let's see the contents of the country and city tables:

`Select * from country;`

`Select * from city;`
Inserting results of a query

Suppose we want to copy data from one table to another . . .

\[
\text{INSERT INTO DEPTS_INFO} \\
(\text{DEPT_NAME, NO_OF_EMPS, TOTAL_SAL}) \\
\text{SELECT} \\
\quad \text{DNAME, COUNT (*)}, \text{SUM(SALARY)} \\
\text{FROM} \\
\quad \text{DEPARTMENT , EMPLOYEE} \\
\text{WHERE} \\
\quad \text{DNUMBER} = \text{DNO} \\
\text{GROUP BY} \\
\quad \text{DNAME ;}
\]

Then insert the result into destination table

First, Write a query that produces the row(s) that you are going to insert.
INSERT INTO DEPTS_INFO
(DEPT_NAME, NO_OF_EMPS, TOTAL_SAL)
SELECT
DNAME, COUNT (*), SUM(SALARY)
FROM
DEPARTMENT, EMPLOYEE
WHERE
DNUMBER = DNO
GROUP BY
DNAME;

Then insert the result into destination table

First, Write a query that produces the row(s) that you are going to insert.
Suppose we want to copy data from one table to another . . .

```sql
INSERT INTO DEPTS_INFO
(DEPT_NAME, NO_OF_EMPS, TOTAL_SAL)
SELECT
  DNAME, COUNT (*), SUM(SALARY)
FROM
  DEPARTMENT , EMPLOYEE
WHERE
  DNUMBER = DNO
GROUP BY
  DNAME ;
```

First, write a query that produces the row(s) that you are going to insert.

Then insert the result into destination table
Update

Used to modify attribute values of one or more selected rows.

Update [table name]
set ...
where ...

Referential integrity

- Updating an attribute value to a value that is not in the domain will not be allowed.
- Updating a foreign key attribute to a value that does not exist in the referencing table will not be allowed.

Lets try it!

Update city and country name
SET-clause specifies the attributes to be modified and their new values

SET  Salary = 1000, manager = "John"
The where clause selects the rows to be modified.

WHERE city="Stockholm"
Referential integrity

Integrity constraints are enforced during update operations

Updating an attribute value to a value that is not in the domain will not be allowed.

- updating population of Sweden to "nine million"

Updating the value of a foreign key attribute to a value that does not exist in the referencing table will not be allowed.

- Updating country column in the city table to a non-existing country.
Update city and country data

1. Change the population of US to 314,500,000

2. In addition, in the recent year, there has been 1% addition to the population of all countries, apply that.

3. By looking at the rows in the city table, American cities have NULL value for the field country, update all of them to "USA".

The update command looks like:

```
Update country
set ... 
where ...
```

At any point you can see the contents of the country and city tables by:

```
Select * from country;
Select * from city;
```
NULL is a very special value!

Each NULL value distinct from other NULL values

Therefore, equality comparison is not appropriate

SQL uses "IS" or "IS NOT" to compare NULLs.
Update city and country data

1. Change the population of US to 314,500,000

2. In addition, in the recent year, there has been 1% addition to the population of all countries, apply that.

3. By looking at the rows in the city table, American cities have NULL value for the field country, update all of them to "USA".

The update command looks like:

```
Update country
set ... where ...
```

At any point you can see the contents of the country and city tables by:

```
Select * from country;
Select * from city;
```
Delete
Removes rows from a table.

DELETE FROM [table name]
WHERE ...

The where clause selects the rows to be deleted.
WHERE country='Sweden'
If no where clause is specified → all rows will be deleted
The where clause selects the rows to be deleted.

WHERE country="Sweden"

if no where clause is specified
--> all rows will be deleted
If we remove a country, what happens to its cities?

Depends on the definition of foreign key:
- If "on delete cascade" is specified, all cities will be removed too.
- Otherwise, the cities will have Null value for country.
Structured Query Language

Note:
This is an introduction in form of a tutorial. For the course & labs more SQL knowledge is needed. Official slides from the book will be available at Studentportalen.
### SELECT * FROM dept;

<table>
<thead>
<tr>
<th>NUMBER</th>
<th>NAME</th>
<th>ID</th>
<th>MANAGER</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Bargain</td>
<td>5</td>
<td>37</td>
</tr>
<tr>
<td>10</td>
<td>Candy</td>
<td>5</td>
<td>13</td>
</tr>
<tr>
<td>14</td>
<td>Jewelry</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>19</td>
<td>Furniture</td>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>30</td>
<td>Home Appliances</td>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>28</td>
<td>Women’s</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>34</td>
<td>Stationary</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>35</td>
<td>Book</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>43</td>
<td>Children’s</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>47</td>
<td>Junior Miss</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>49</td>
<td>Toys</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>58</td>
<td>Men’s</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>60</td>
<td>Sportswear</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>63</td>
<td>Women’s</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>65</td>
<td>Junior’s</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>70</td>
<td>Women’s</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>73</td>
<td>Children’s</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>99</td>
<td>Giftwrap</td>
<td>5</td>
<td>1</td>
</tr>
</tbody>
</table>

**Select * from dept;**

Shows all the whole table, that is

- all column
- all rows
What if we are interested in a subset of attributes?

i.e. some columns?

<table>
<thead>
<tr>
<th>NUMBER</th>
<th>NAME</th>
<th>STORE</th>
<th>FLOOR</th>
<th>MANAGER</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Bargain</td>
<td>5</td>
<td>0</td>
<td>37</td>
</tr>
<tr>
<td>10</td>
<td>Candy</td>
<td>5</td>
<td>1</td>
<td>13</td>
</tr>
<tr>
<td>14</td>
<td>Jewelry</td>
<td>8</td>
<td>1</td>
<td>33</td>
</tr>
<tr>
<td>19</td>
<td>Furniture</td>
<td>7</td>
<td>4</td>
<td>26</td>
</tr>
<tr>
<td>20</td>
<td>Major Appliances</td>
<td>7</td>
<td>4</td>
<td>26</td>
</tr>
<tr>
<td>26</td>
<td>Linens</td>
<td>7</td>
<td>3</td>
<td>157</td>
</tr>
<tr>
<td>28</td>
<td>Women’s</td>
<td>8</td>
<td>2</td>
<td>32</td>
</tr>
<tr>
<td>34</td>
<td>Stationary</td>
<td>5</td>
<td>1</td>
<td>33</td>
</tr>
<tr>
<td>35</td>
<td>Book</td>
<td>5</td>
<td>1</td>
<td>55</td>
</tr>
<tr>
<td>43</td>
<td>Children’s</td>
<td>8</td>
<td>2</td>
<td>32</td>
</tr>
<tr>
<td>47</td>
<td>Junior Miss</td>
<td>7</td>
<td>2</td>
<td>129</td>
</tr>
<tr>
<td>49</td>
<td>Toys</td>
<td>8</td>
<td>2</td>
<td>35</td>
</tr>
<tr>
<td>58</td>
<td>Men’s</td>
<td>7</td>
<td>2</td>
<td>129</td>
</tr>
<tr>
<td>60</td>
<td>Sportswear</td>
<td>5</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>63</td>
<td>Women’s</td>
<td>7</td>
<td>3</td>
<td>37</td>
</tr>
<tr>
<td>65</td>
<td>Junior’s</td>
<td>7</td>
<td>3</td>
<td>10</td>
</tr>
<tr>
<td>70</td>
<td>Women’s</td>
<td>5</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>73</td>
<td>Children’s</td>
<td>5</td>
<td>1</td>
<td>98</td>
</tr>
<tr>
<td>99</td>
<td>Giftwrap</td>
<td>5</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Select name, store
from dept;

- Returns only name and store columns

Select name
from dept;

- Returns only the name column.

Select distinct name
from dept;

- Removes the duplicates
What if we are interested in some rows?

```
SELECT * FROM dept;
NUMBER  NAME          STORE FLOOR MANAGER
-------- ----------- ------- ------ -------
     1  Bargain      5      0     37
     10 Candy       5      1     13
     14 Jewelry     8      1     33
     19 Furniture   7      4     26
     20 Major Appliances 7      4     26
     26 Linens     7      3    157
     28 Women’s    8      2     32
     34 Stationary 5      1     33
     35 Book       5      1     55
     43 Children’s 8      2     32
     47 Junior Miss 7      2    129
     49 Toys       8      2     35
     58 Men’s      7      2    129
     60 Sportswear 5      1     10
     63 Women’s    7      3     32
     65 Junior’s   7      3     37
     70 Women’s    5      1     10
     73 Children’s 5      1     10
     99 Giftwrap   5      1     98
```

The "where" clause is used to specify the filtering conditions.
- can be a combination of conditions using and/or constructs.

Retrieve name of all departments that are located in floor 1 and are managed by manager number 10.

```
Select ?
from ?
Where ? and ? ;
```
Substring comparison

The *LIKE* comparison operator is used to compare partial strings.
- is part of the where clause

Two reserved characters are used:
- '%' replaces an arbitrary number of characters.
- '_' replaces a single arbitrary character.

Example: retrieving first name and last name of all employees living in Houston, TX

```
Select f_name, l_name
from employee
Where address like '%Houston, TX%';
```
Let's try Substring matching

```
SELECT * FROM item;
```

<table>
<thead>
<tr>
<th>NUMBER</th>
<th>NAME</th>
<th>PRICE</th>
<th>SUPPLIER</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>Wash Cloth</td>
<td>75</td>
<td>213</td>
</tr>
<tr>
<td>19</td>
<td>Bellbottoms</td>
<td>450</td>
<td>33</td>
</tr>
<tr>
<td>21</td>
<td>ABC Blocks</td>
<td>198</td>
<td>125</td>
</tr>
<tr>
<td>23</td>
<td>1 lb Box</td>
<td>215</td>
<td>42</td>
</tr>
<tr>
<td>25</td>
<td>2 lb Box, Mix</td>
<td>450</td>
<td>42</td>
</tr>
<tr>
<td>26</td>
<td>Earrings</td>
<td>1000</td>
<td>199</td>
</tr>
<tr>
<td>43</td>
<td>Maze</td>
<td>325</td>
<td>89</td>
</tr>
<tr>
<td>52</td>
<td>Jacket</td>
<td>3295</td>
<td>15</td>
</tr>
<tr>
<td>101</td>
<td>Slacks</td>
<td>1600</td>
<td>15</td>
</tr>
<tr>
<td>106</td>
<td>Clock Book</td>
<td>198</td>
<td>125</td>
</tr>
<tr>
<td>107</td>
<td>The 'Feel' Book</td>
<td>225</td>
<td>89</td>
</tr>
<tr>
<td>115</td>
<td>Gold Ring</td>
<td>4995</td>
<td>199</td>
</tr>
<tr>
<td>118</td>
<td>Towels, Bath</td>
<td>250</td>
<td>213</td>
</tr>
<tr>
<td>119</td>
<td>Squeeze Ball</td>
<td>250</td>
<td>89</td>
</tr>
<tr>
<td>120</td>
<td>Twin Sheet</td>
<td>800</td>
<td>213</td>
</tr>
<tr>
<td>121</td>
<td>Queen Sheet</td>
<td>1375</td>
<td>213</td>
</tr>
<tr>
<td>127</td>
<td>Ski Jumpsuit</td>
<td>4350</td>
<td>15</td>
</tr>
<tr>
<td>165</td>
<td>Jean</td>
<td>825</td>
<td>33</td>
</tr>
<tr>
<td>258</td>
<td>Shirt</td>
<td>650</td>
<td>33</td>
</tr>
<tr>
<td>301</td>
<td>Boy's Jean Suit</td>
<td>1250</td>
<td>33</td>
</tr>
</tbody>
</table>

What are the names and prices of books?

```
Select ??
from item
Where ?
```

The previous example was:

```
Select f_name,l_name
from employee
Where address like '%Houston,TX%';
```

20 rows found
"Simple" queries

Queries that retrieve information from one table.

```
SELECT <attribute list>
FROM <table>
[WHERE <condition>]
```
What if the query involves more than one table? like in . . .

Retrieve store numbers and their corresponding state. here we need to "join" tables store and city.
Cartesian product

Select *
from A, B

the comma produces the Cartesian product
Cartesian product

Select * from A, B

the comma produces the Cartesian product
The Cartesian product of A and B provides all possible combination of rows in the tables A and B.

A natural join, denoted by \( \Join \), joins two tables such that the common attribute has the same value. Common attribute names are frequent in tables that are related by primary-foreign keys.

A natural join includes a vertical filtering to provide only one copy of the common attribute.
Join condition

Is a filter specified in the where clause that is applied to the Cartesian product of A and B.

Specifies what rows are "related".
A and B joined

Join condition

The Cartesian product of A and B

Cartesian product

Select *
from A, B

Table A

Table B
Let's join store and city tables

```
SELECT * FROM store;
NUMBER CITY
-------------
 5 San Francisco
 7 Oakland
 8 El Cerrito
3 rows found
```

Retrieve store numbers and their corresponding state.

First form the Cartesian product of store and city by:

```
Select * from store, city;
```

What is the join condition?

In most cases, an equality on the foreign key to its referencing key.

```
Select * from store, city
where city=name;
```

Retrieve store numbers and their corresponding state.

First form the Cartesian product of store and city by:

Select * from store, city;

What is the join condition?

In most cases, an equality on the foreign key to its referencing key.

Select * from store, city where city=name;
There are explicit join operators in SQL.

Select * from store, city where city=name;
is equal to:
Select * from store join city on city=name;

There is more into Join

Read from the book:
- Left outer join
- Right outer join
- Full outer join (AKA cross join)
There is more into Join

Read from the book:
• Left outer join
• Right outer join
• Full outer join
  (AKA cross join)
Recall the country table we added to the database . . .

Write a query that returns name of cities and their official language.

```sql
select name, language
from city, country
where country = name;
```

*The above query is ambiguous since both tables city and country have an attribute called "name".*

```sql
select city.name as city_name, language
from city, country c
where country = c.name;
```

*C is an Alias for table country, city_name is an alias for city.name*
both tables city and country have an attribute called "name".

select city.name as city_name, language from city, country c
where country = c.name;

C is an Alias for table country, city_name is an alias for city.name
Aliases are needed to join a table to itself

```
SELECT * FROM employee;

<table>
<thead>
<tr>
<th>NUMBER</th>
<th>NAME</th>
<th>SALARY</th>
<th>MANAGER</th>
<th>BIRTHYEAR</th>
<th>STARTYEAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Ross, Stanley</td>
<td>15908</td>
<td>199</td>
<td>1927</td>
<td>1945</td>
</tr>
<tr>
<td>11</td>
<td>Ross, Stuart</td>
<td>12067</td>
<td>-</td>
<td>1931</td>
<td>1932</td>
</tr>
<tr>
<td>13</td>
<td>Edwards, Peter</td>
<td>9000</td>
<td>199</td>
<td>1928</td>
<td>1958</td>
</tr>
<tr>
<td>26</td>
<td>Thompson, Bob</td>
<td>13000</td>
<td>199</td>
<td>1930</td>
<td>1970</td>
</tr>
<tr>
<td>32</td>
<td>Smythe, Carol</td>
<td>9050</td>
<td>199</td>
<td>1929</td>
<td>1967</td>
</tr>
<tr>
<td>33</td>
<td>Hayes, Evelyn</td>
<td>10100</td>
<td>199</td>
<td>1931</td>
<td>1963</td>
</tr>
<tr>
<td>35</td>
<td>Evans, Michael</td>
<td>5000</td>
<td>32</td>
<td>1952</td>
<td>1974</td>
</tr>
<tr>
<td>37</td>
<td>Raveen, Lemont</td>
<td>11985</td>
<td>26</td>
<td>1950</td>
<td>1974</td>
</tr>
<tr>
<td>55</td>
<td>James, Mary</td>
<td>12000</td>
<td>199</td>
<td>1920</td>
<td>1969</td>
</tr>
<tr>
<td>98</td>
<td>Williams, Judy</td>
<td>9000</td>
<td>199</td>
<td>1935</td>
<td>1969</td>
</tr>
<tr>
<td>129</td>
<td>Thomas, Tom</td>
<td>10000</td>
<td>199</td>
<td>1941</td>
<td>1962</td>
</tr>
<tr>
<td>157</td>
<td>Jones, Tim</td>
<td>12000</td>
<td>199</td>
<td>1940</td>
<td>1960</td>
</tr>
<tr>
<td>199</td>
<td>Bullock, J.D.</td>
<td>27000</td>
<td>-</td>
<td>1920</td>
<td>1920</td>
</tr>
<tr>
<td>215</td>
<td>Collins, Joanne</td>
<td>7000</td>
<td>10</td>
<td>1950</td>
<td>1971</td>
</tr>
<tr>
<td>430</td>
<td>Brunet, Paul C.</td>
<td>17674</td>
<td>129</td>
<td>1938</td>
<td>1959</td>
</tr>
<tr>
<td>843</td>
<td>Schmidt, Herman</td>
<td>11204</td>
<td>26</td>
<td>1936</td>
<td>1956</td>
</tr>
<tr>
<td>994</td>
<td>Iwano, Masahiro</td>
<td>15641</td>
<td>129</td>
<td>1944</td>
<td>1970</td>
</tr>
<tr>
<td>1110</td>
<td>Smith, Paul</td>
<td>6000</td>
<td>33</td>
<td>1952</td>
<td>1973</td>
</tr>
<tr>
<td>1380</td>
<td>Unstad, Richard</td>
<td>8779</td>
<td>13</td>
<td>1952</td>
<td>1971</td>
</tr>
<tr>
<td>1523</td>
<td>Zugnoni, Arthur A.</td>
<td>19868</td>
<td>129</td>
<td>1928</td>
<td>1949</td>
</tr>
<tr>
<td>1639</td>
<td>Choy, Wanda</td>
<td>11160</td>
<td>55</td>
<td>1947</td>
<td>1970</td>
</tr>
<tr>
<td>2398</td>
<td>Wallace, Maggie J.</td>
<td>7880</td>
<td>26</td>
<td>1940</td>
<td>1959</td>
</tr>
<tr>
<td>4901</td>
<td>Bailey, Chas M.</td>
<td>8377</td>
<td>32</td>
<td>1956</td>
<td>1975</td>
</tr>
<tr>
<td>5119</td>
<td>Bono, Sonny</td>
<td>13621</td>
<td>55</td>
<td>1939</td>
<td>1963</td>
</tr>
<tr>
<td>5219</td>
<td>Schwarz, Jason B.</td>
<td>13374</td>
<td>33</td>
<td>1944</td>
<td>1959</td>
</tr>
</tbody>
</table>

25 rows found
```

Retrieve manager name for all employees that have a manager.

```
Select e.name, m.name from employee e,
employee m where e.manager = m.number;
```
Alias

Aliases are needed for:
- Readability and clear semantics
- Resolving attribute ambiguity.
- Having several instances of the same table/attribute.
Joining tables

When you need to combine information from more than one table.
### SELECT * FROM employee;

<table>
<thead>
<tr>
<th>NUMBER</th>
<th>NAME</th>
<th>SALARY</th>
<th>MANAGER</th>
<th>BIRTHYEAR</th>
<th>STARTYEAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Ross, Stanley</td>
<td>15908</td>
<td>199</td>
<td>1927</td>
<td>1945</td>
</tr>
<tr>
<td>11</td>
<td>Ross, Stuart</td>
<td>12067</td>
<td>-</td>
<td>1931</td>
<td>1932</td>
</tr>
<tr>
<td>13</td>
<td>Edwards, Peter</td>
<td>9000</td>
<td>199</td>
<td>1928</td>
<td>1958</td>
</tr>
<tr>
<td>26</td>
<td>Thompson, Bob</td>
<td>13000</td>
<td>199</td>
<td>1930</td>
<td>1970</td>
</tr>
<tr>
<td>32</td>
<td>Smythe, Carol</td>
<td>9050</td>
<td>199</td>
<td>1929</td>
<td>1967</td>
</tr>
<tr>
<td>33</td>
<td>Hayes, Evelyn</td>
<td>10100</td>
<td>199</td>
<td>1931</td>
<td>1963</td>
</tr>
<tr>
<td>35</td>
<td>Evans, Michael</td>
<td>5000</td>
<td>32</td>
<td>1952</td>
<td>1974</td>
</tr>
<tr>
<td>37</td>
<td>Raveen, Lemont</td>
<td>11985</td>
<td>26</td>
<td>1950</td>
<td>1974</td>
</tr>
<tr>
<td>55</td>
<td>James, Mary</td>
<td>12000</td>
<td>199</td>
<td>1920</td>
<td>1969</td>
</tr>
<tr>
<td>98</td>
<td>Williams, Judy</td>
<td>9000</td>
<td>199</td>
<td>1935</td>
<td>1969</td>
</tr>
<tr>
<td>129</td>
<td>Thomas, Tom</td>
<td>10000</td>
<td>199</td>
<td>1941</td>
<td>1982</td>
</tr>
<tr>
<td>157</td>
<td>Jones, Tim</td>
<td>12000</td>
<td>199</td>
<td>1940</td>
<td>1960</td>
</tr>
<tr>
<td>199</td>
<td>Bullock, J.D.</td>
<td>27000</td>
<td>-</td>
<td>1920</td>
<td>1920</td>
</tr>
<tr>
<td>215</td>
<td>Collins, Joanne</td>
<td>7000</td>
<td>10</td>
<td>1950</td>
<td>1971</td>
</tr>
<tr>
<td>430</td>
<td>Brunet, Paul C.</td>
<td>17674</td>
<td>129</td>
<td>1938</td>
<td>1959</td>
</tr>
<tr>
<td>843</td>
<td>Schmidt, Herman</td>
<td>11204</td>
<td>26</td>
<td>1936</td>
<td>1966</td>
</tr>
<tr>
<td>994</td>
<td>Iwano, Masahiro</td>
<td>15641</td>
<td>129</td>
<td>1944</td>
<td>1970</td>
</tr>
<tr>
<td>1110</td>
<td>Smith, Paul</td>
<td>6000</td>
<td>33</td>
<td>1952</td>
<td>1973</td>
</tr>
<tr>
<td>1330</td>
<td>Unstad, Richard</td>
<td>8779</td>
<td>13</td>
<td>1952</td>
<td>1971</td>
</tr>
<tr>
<td>1523</td>
<td>Zugnoni, Arthur A.</td>
<td>19888</td>
<td>129</td>
<td>1928</td>
<td>1949</td>
</tr>
<tr>
<td>1639</td>
<td>Choy, Wanda</td>
<td>11160</td>
<td>55</td>
<td>1947</td>
<td>1970</td>
</tr>
<tr>
<td>2398</td>
<td>Wallace, Maggie J.</td>
<td>7880</td>
<td>26</td>
<td>1940</td>
<td>1959</td>
</tr>
<tr>
<td>4901</td>
<td>Bailey, Chas M.</td>
<td>8377</td>
<td>32</td>
<td>1956</td>
<td>1975</td>
</tr>
<tr>
<td>5119</td>
<td>Bono, Sonny</td>
<td>13621</td>
<td>55</td>
<td>1939</td>
<td>1963</td>
</tr>
<tr>
<td>5219</td>
<td>Schwarz, Jason B.</td>
<td>13374</td>
<td>33</td>
<td>1944</td>
<td>1959</td>
</tr>
</tbody>
</table>

25 rows found

---

**What is the average salary of an employee?**

```
select avg(salary) from employee;
```

**How many managers are there in the company?**

```
select count(manager) from employee;
```

returns 23! why?

```
select count(distinct manager) from employee;
```
Aggregate functions

Operate on the set of values of a column of a relation, and return a value:

• avg : average value
• min : minimum value
• max : maximum value
• sum : sum of values
• count : number of values

In the previous example, they operated on the whole table, but ...
group by

... but sometimes we want to break down statistics into groups:

```
SELECT * FROM employee;
```

<table>
<thead>
<tr>
<th>NUMBER</th>
<th>NAME</th>
<th>SALARY</th>
<th>MANAGER</th>
<th>BIRTHYEAR</th>
<th>STARTYEAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Ross, Stanley</td>
<td>15908</td>
<td>199</td>
<td>1927</td>
<td>1945</td>
</tr>
<tr>
<td>11</td>
<td>Ross, Stuart</td>
<td>12067</td>
<td>-</td>
<td>1931</td>
<td>1932</td>
</tr>
<tr>
<td>13</td>
<td>Edwards, Peter</td>
<td>9000</td>
<td>199</td>
<td>1928</td>
<td>1958</td>
</tr>
<tr>
<td>26</td>
<td>Thompson, Bob</td>
<td>13000</td>
<td>199</td>
<td>1930</td>
<td>1970</td>
</tr>
<tr>
<td>32</td>
<td>Smythe, Carol</td>
<td>9050</td>
<td>199</td>
<td>1929</td>
<td>1967</td>
</tr>
<tr>
<td>33</td>
<td>Hayes, Evelyn</td>
<td>10100</td>
<td>199</td>
<td>1931</td>
<td>1963</td>
</tr>
<tr>
<td>35</td>
<td>Evans, Michael</td>
<td>5000</td>
<td>32</td>
<td>1952</td>
<td>1974</td>
</tr>
<tr>
<td>37</td>
<td>Raveen, Lemont</td>
<td>11985</td>
<td>26</td>
<td>1950</td>
<td>1974</td>
</tr>
<tr>
<td>55</td>
<td>James, Mary</td>
<td>12000</td>
<td>199</td>
<td>1920</td>
<td>1969</td>
</tr>
<tr>
<td>98</td>
<td>Williams, Judy</td>
<td>9000</td>
<td>199</td>
<td>1935</td>
<td>1969</td>
</tr>
<tr>
<td>129</td>
<td>Thomas, Tom</td>
<td>10000</td>
<td>199</td>
<td>1941</td>
<td>1962</td>
</tr>
<tr>
<td>157</td>
<td>Jones, Tim</td>
<td>12000</td>
<td>199</td>
<td>1940</td>
<td>1960</td>
</tr>
<tr>
<td>199</td>
<td>Bullock, J.D.</td>
<td>27000</td>
<td>-</td>
<td>1920</td>
<td>1920</td>
</tr>
<tr>
<td>215</td>
<td>Collins, Joanne</td>
<td>7000</td>
<td>10</td>
<td>1950</td>
<td>1971</td>
</tr>
<tr>
<td>430</td>
<td>Brunet, Paul C.</td>
<td>17674</td>
<td>129</td>
<td>1938</td>
<td>1959</td>
</tr>
<tr>
<td>843</td>
<td>Schmidt, Herman</td>
<td>11204</td>
<td>26</td>
<td>1936</td>
<td>1956</td>
</tr>
<tr>
<td>994</td>
<td>Iwano, Masahiro</td>
<td>15641</td>
<td>129</td>
<td>1944</td>
<td>1970</td>
</tr>
<tr>
<td>1110</td>
<td>Smith, Paul</td>
<td>6000</td>
<td>33</td>
<td>1952</td>
<td>1973</td>
</tr>
<tr>
<td>1330</td>
<td>Onstad, Richard</td>
<td>8779</td>
<td>13</td>
<td>1952</td>
<td>1971</td>
</tr>
<tr>
<td>1523</td>
<td>Zugenoni, Arthur A.</td>
<td>19868</td>
<td>129</td>
<td>1928</td>
<td>1949</td>
</tr>
<tr>
<td>1639</td>
<td>Choy, Wanda</td>
<td>11160</td>
<td>55</td>
<td>1947</td>
<td>1970</td>
</tr>
<tr>
<td>2398</td>
<td>Wallace, Maggie J.</td>
<td>7880</td>
<td>26</td>
<td>1940</td>
<td>1959</td>
</tr>
<tr>
<td>4901</td>
<td>Bailey, Chas M.</td>
<td>8377</td>
<td>32</td>
<td>1956</td>
<td>1975</td>
</tr>
<tr>
<td>5119</td>
<td>Bono, Sonny</td>
<td>13621</td>
<td>55</td>
<td>1939</td>
<td>1963</td>
</tr>
<tr>
<td>5219</td>
<td>Schwarz, Jason B.</td>
<td>13374</td>
<td>33</td>
<td>1944</td>
<td>1959</td>
</tr>
</tbody>
</table>

25 rows found

How many employees are managed by each manager?

```
Select manager, count(*) from employee
group by manager;
```

More than one attribute can be specified in a group by clause, BUT:

attributes list in the select list (except aggregate functions) should be always present in the group by.
Aggregate/group-by/join

Group by can be applied to any relation/table. It is very common that the relation itself is produced by joining some other relations.

```
SELECT * FROM store;
NUMBER  CITY
========  ========
  5  San Francisco
  7   Oakland
  8    El Cerrito
```

3 rows found

How many departments exist per city?

If we had a column city next to each row, we could apply the count() and answer the query.

1. Write a join query that "brings in" the city into the dept;
2. Add the aggregate/group by clauses to answer the query.

```
SELECT city, count(*)
FROM store s, dept d
WHERE s.number=d.store
GROUP BY city;
```
1. `select * from store s, dept d where s.number=d.store;`

2. `select city, count(*) from store s, dept d where s.number=d.store group by city;`
Aggregate functions

for calculating statistics:

- count, min, avg, ...
- Usually applied to groups of rows
- Many times together with join
Set operations

- There is a union operation (UNION), and in some versions of SQL there are set difference (MINUS) and intersection (INTERSECT) operations.

- The resulting relations of these set operations are sets of tuples; duplicate tuples are eliminated from the result.

- The set operations apply only to union compatible relations; the two relations must have the same attributes and the attributes must appear in the same order.
An example

```
SELECT * FROM dept;
```

<table>
<thead>
<tr>
<th>NUMBER</th>
<th>NAME</th>
<th>STORE</th>
<th>FLOOR</th>
<th>MANAGER</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Bargain</td>
<td>5</td>
<td>0</td>
<td>37</td>
</tr>
<tr>
<td>10</td>
<td>Candy</td>
<td>5</td>
<td>1</td>
<td>13</td>
</tr>
<tr>
<td>14</td>
<td>Jewelry</td>
<td>8</td>
<td>1</td>
<td>33</td>
</tr>
<tr>
<td>19</td>
<td>Furniture</td>
<td>7</td>
<td>4</td>
<td>26</td>
</tr>
<tr>
<td>20</td>
<td>Major Appliances</td>
<td>7</td>
<td>4</td>
<td>26</td>
</tr>
<tr>
<td>26</td>
<td>Linens</td>
<td>7</td>
<td>3</td>
<td>157</td>
</tr>
<tr>
<td>28</td>
<td>Women’s</td>
<td>8</td>
<td>2</td>
<td>32</td>
</tr>
<tr>
<td>34</td>
<td>Stationary</td>
<td>5</td>
<td>1</td>
<td>33</td>
</tr>
<tr>
<td>35</td>
<td>Book</td>
<td>5</td>
<td>1</td>
<td>55</td>
</tr>
<tr>
<td>43</td>
<td>Children’s</td>
<td>8</td>
<td>2</td>
<td>32</td>
</tr>
<tr>
<td>47</td>
<td>Junior Miss</td>
<td>7</td>
<td>2</td>
<td>129</td>
</tr>
<tr>
<td>49</td>
<td>Toys</td>
<td>8</td>
<td>2</td>
<td>35</td>
</tr>
<tr>
<td>58</td>
<td>Men’s</td>
<td>7</td>
<td>2</td>
<td>129</td>
</tr>
<tr>
<td>60</td>
<td>Sportswear</td>
<td>5</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>63</td>
<td>Women’s</td>
<td>7</td>
<td>3</td>
<td>32</td>
</tr>
<tr>
<td>65</td>
<td>Junior’s</td>
<td>7</td>
<td>3</td>
<td>37</td>
</tr>
<tr>
<td>70</td>
<td>Women’s</td>
<td>5</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>73</td>
<td>Children’s</td>
<td>5</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>99</td>
<td>Giftwrap</td>
<td>5</td>
<td>1</td>
<td>98</td>
</tr>
</tbody>
</table>

list of departments in floors 1 or 3:

(Select name from dept where floor=3) union (Select name from dept where floor=1);

What is the difference between this and the following query:

Select name from dept where floor=3 or floor=1;

Union removes duplicates
```
SELECT * FROM store;
  NUMBER CITY
=---------- =----------
   5 San Francisco
   7 Oakland
   8 El Cerrito

3 rows found
```

```
SELECT * FROM dept;
  NUMBER NAME
=---------- =----------
   1 Bargain
   5 5 Candy
  10 1 Jewelry
  14 1 33
  19 4 26
  20 4 26
  26 3 157
  28 2 32
  34 1 33
  35 1 55
  43 2 32
  47 2 129
  49 2 35
  58 2 129
  60 1 10
  63 3 32
  65 3 37
  70 1 10
  73 1 10
  99 1 98

```

**Select name**

**from dept**

**Where store in** (select number from store where city='Oakland')

Retrieve the name of departments in Oakland.

A join could be used instead. [home work: what would it be?]
7 Oakland
8 El Cerrito

3 rows found

Select name from dept
Where store in (select number from store where city='Oakland')

; Retrieve the name of department
A join could be used instead. [how would it be?]
Nested queries

Select name from dept
Where store in (select number from store where city='Oakland')
;
Retrieve the name of departments in Oakland.

A join could be used instead. [home work: what would it be?]

- Many queries can be expressed in several ways read more in additional slides available from Studentportal.
Queries

SELECT <attribute list> 
FROM <table list> 
[WHERE <condition>] 
[GROUP BY <grouping attribute(s)>] 
[HAVING <group condition>] 
[ORDER BY <attribute list>]

Optional

"Simple" queries
Queries that retrieve information from one table
SELECT columns FROM table_name [WHERE condition]

Joining tables
When two or more related tables need to be combined
SELECT columns FROM table1 JOIN table2 ON table1.key = table2.key

Aggregate functions
Functions that summarize data
COUNT, SUM, AVG, MAX, MIN

Set operations
Carrying out set operations on results
INTERSECT, UNION, EXCEPT

Nested queries
Subqueries that are embedded within another query
(SELECT ... FROM ... WHERE (SELECT ... FROM ...))
Structured Query Language

Queries

Data Definition Language
The subset of SQL that is used in creating, changing, and deleting the description (schema) of tables.

Data Manipulation Language
The subset of SQL used in adding, modifying, and deleting rows in the table.

Note:
This is an introduction in form of a tutorial. For the course and labs more SQL knowledge is needed. Official slides from the book will be available at Studentportalen.