DATABASE SYSTEMS

SQL fundamentals

Bence Molnár

2022.04.12

BUDAPESTI MŰSZAKI
ÉS GAZDASÁGTUDOMÁNYI EGYETEM
Építőmérnöki Kar - építőmérnöki képzés 1782 óta

Fotogrammetria és Térinformatika Tanszék

Agenda

- Fundamentals
- Analytic operations
- Data definition
- Data manipulation

SQL

BUDAPESTI MŰSZAKI
ÉS GAZDASÁGTUDOMÁNYI EGYETEM
Építőmérnöki Kar - építőmérnöki képzés 1782 óta

Fotogrammetria és Térinformatika Tanszék

History

- 1970s: IBM SEQUEL (Structured English QUery Language)
- Structured/Standard Query Language
- 1986: ANSI standard, 1987: ISO standard
- SQL2 ('92), SQL3 ('99), ...
- Continuous improvement (SQL2011)
- More or less all DBMS supports this standard; concept is common
- Relational Software, Inc. → Oracle Corp.

Fundamentals

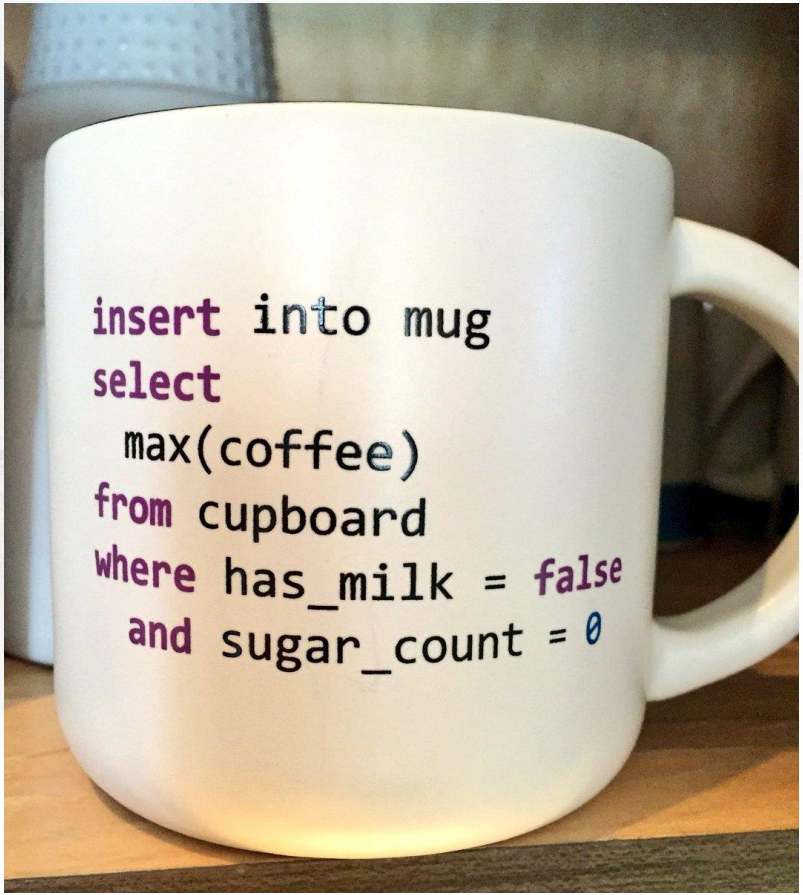
- Relies on relational algebra
- Non-procedural language (declarative); this means we need to formulate commands to computer in the way what to do and not how to do.
- Data definition (DDL) -, Data manipulation (DML) -, Data control (DCL) -, Query (QL) language

SQL BASICS

Textual description of operations (commands)

Need to follow strict rules

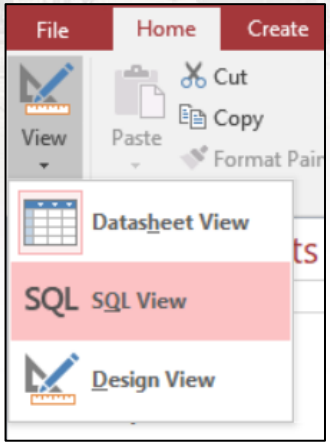
- Given set of commands
- Sentence (command) ends with “;”
- Order of command is given


insert into mug
select
max(coffee)
from cupboard
where has_milk = false
and sugar_count = 0

Access

A graphically (in Design view) defined query is automatically translated and displayed to SQL.

A SQL defined query not necessarily transformed to a graphical query!

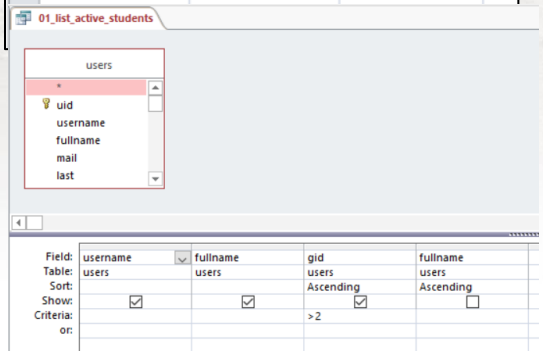


Example – I.

List all active students and order based on their names!

01_list_active_students

username	fullname	gid
cf99d2	-	Egyéni
cf9bad	-	Egyéni
cf9bbc	-	Szerkezet
cf9b16	-	Szerkezet
cf9b19	-	Szerkezet
cf9b1d	-	Szerkezet

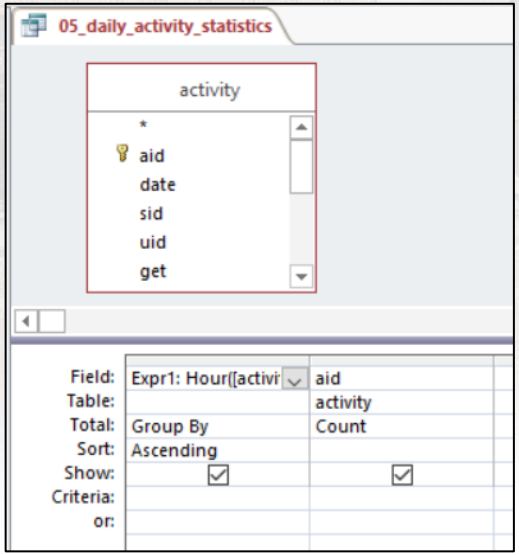


01_list_active_students

```
SELECT users.username, users.fullname, users.gid
FROM users
WHERE (((users.gid)>2))
ORDER BY users.gid, users.fullname;
```

Example – II.

Show activity statistics within a day grouped by hours.



05_daily_activity_statistics

```
SELECT Hour([activity].[date]) AS Hours, Count(activity.aid) AS CountOfaid
FROM activity
GROUP BY Hour([activity].[date])
ORDER BY Hour([activity].[date]);
```

Analytic operations

Queries

BUDAPESTI MŰSZAKI
ÉS GAZDASÁGTUDOMÁNYI EGYETEM
Építőmérnöki Kar - építőmérnöki képzés 1782 óta

Fotogrammetria és Térinformatika Tanszék

TRANSLATE FROM RELATIONAL ALGEBRA

Relációs algebra	SQL
R	`SELECT * FROM r;`
$\pi_{t \to tt}(R)$	`SELECT t AS tt FROM r;`
$\sigma_{felt}(R)$	`SELECT * FROM r WHERE felt;`
$\pi_t(\sigma_{felt}(R))$	`SELECT t FROM r WHERE felt;`
$R \cup S$	`SELECT * FROM r UNION SELECT * FROM s;`
$R \cap S$	`SELECT * FROM r INTERSECT SELECT * FROM s;`
$R \backslash S$	`SELECT * FROM r EXCEPT SELECT * FROM s;`
$R \times S$	`SELECT * FROM r, s; (or CROSS JOIN)`
$R \bowtie S$	`SELECT * FROM r NATURAL JOIN s;`
$R \bowtie_{felt} S$	`SELECT * FROM r JOIN s ON felt;`
$\delta(R)$	`SELECT DISTINCT * FROM r;`
$\gamma_t(R)$	`SELECT t FROM r GROUP BY t;`
$\tau_t(R)$	`SELECT * FROM r ORDER BY t;`

SELECT QUERY

SELECT command never performs any modification on data, does not changes table structure (column names), but displays (temporally) the result based on current data.

Viewing a query means a new execution on actual dataset.

The result of SELECT is also a relation which can be reused in other queries.

Projection

Projection [π]: we select the columns to display and all other will be hided; the result is a relation.

SQL: `SELECT attr1, attr2, … FROM r;`

- `SELECT name, year_of_birth FROM student;`

neptun	name	year_of_birth

- `SELECT * FROM student;`

neptun	name	year_of_birth

Projection

- `SELECT name FROM student;`

π_{name}(

name	grade
John Doe	2
Joe Average	3
Peter Common	5
Jane Doe	3

)=

name
John Doe
Joe Average
Peter Common
Jane Doe

- `SELECT name, grade FROM student;`

$\pi_{\text{name},grade}$(

name	grade	presence
John Doe	2	14
Joe Average	3	14
Peter Common	5	13
Jane Doe	3	10

)=

name	grade
John Doe	2
Joe Average	3
Peter Common	5
Jane Doe	3

Selection (filtering)

Filter the rows

SQL: `SELECT * FROM S WHERE attr1 comp1 value1 op1 attr2 comp2 value2…;`

compX ∈ ('=', '<', '>', '≠', '≤', '≥')

opX ∈ ('AND', 'OR', 'XOR', 'NOT'))

- `SELECT * FROM student WHERE name = 'John Doe';`

We can define multiple criteria, use logical operators to combine them!

This command has a special importance at data manipulation commands

Selection

- `SELECT * FROM student WHERE grade=3;`

$\sigma_{grade=3}$ (

name	grade
John Doe	2
Joe Average	3
Peter Common	5
John Doe	3

)=

name	grade
Joe Average	3
John Doe	3

- `SELECT * FROM student WHERE grade>1 AND presence>10;`

$\sigma_{grade>1\ AND\ presence>10}$ (

name	grade	presence
John Doe	1	14
Joe Average	3	14
Peter Common	5	13
Jane Doe	3	10

)=?

Functions for calculations

- `SELECT AVG(grade) FROM student;`
- `SELECT MIN(grade) FROM student;`
- `SELECT MAX(grade) FROM student;`
- `SELECT COUNT(*) FROM student;`
- `SELECT SUM(grade)/COUNT(*) FROM student WHERE grade IS NOT NULL;`

Substring comparison (pattern based)

```
…attribute LIKE 'sampletext'…
```

- `SELECT * FROM student WHERE name LIKE 'John*';`

Wildcard characters:

- *'*' any character; 0,1,any length (MySQL: %)*
- *'?' any single alphabetic character (MySQL: _)*
- *'#' any single numeric character*

TEXT FUNCTIONS

- `SELECT firstname & ' ' & lastname FROM student;`
- `SELECT * FROM student WHERE firstname & ' ' & lastname = 'John Doe';`

Access: &

MySQL (MariaDB): CONCAT()

PgSQL: ||

Complex queries

- `SELECT name FROM student WHERE (name LIKE '*Doe' OR name LIKE '*Common') AND name LIKE 'John*';`
- `SELECT name FROM student WHERE name LIKE 'P?ter*';`
- `SELECT length_of_presentation*60 AS minute FROM subjects;`
- `SELECT subjectname FROM subject WHERE length_of_presenttion*num_of_presentations>credit;`
- `SELECT name FROM student WHERE subject = 'Databases' AND NOT failed;`

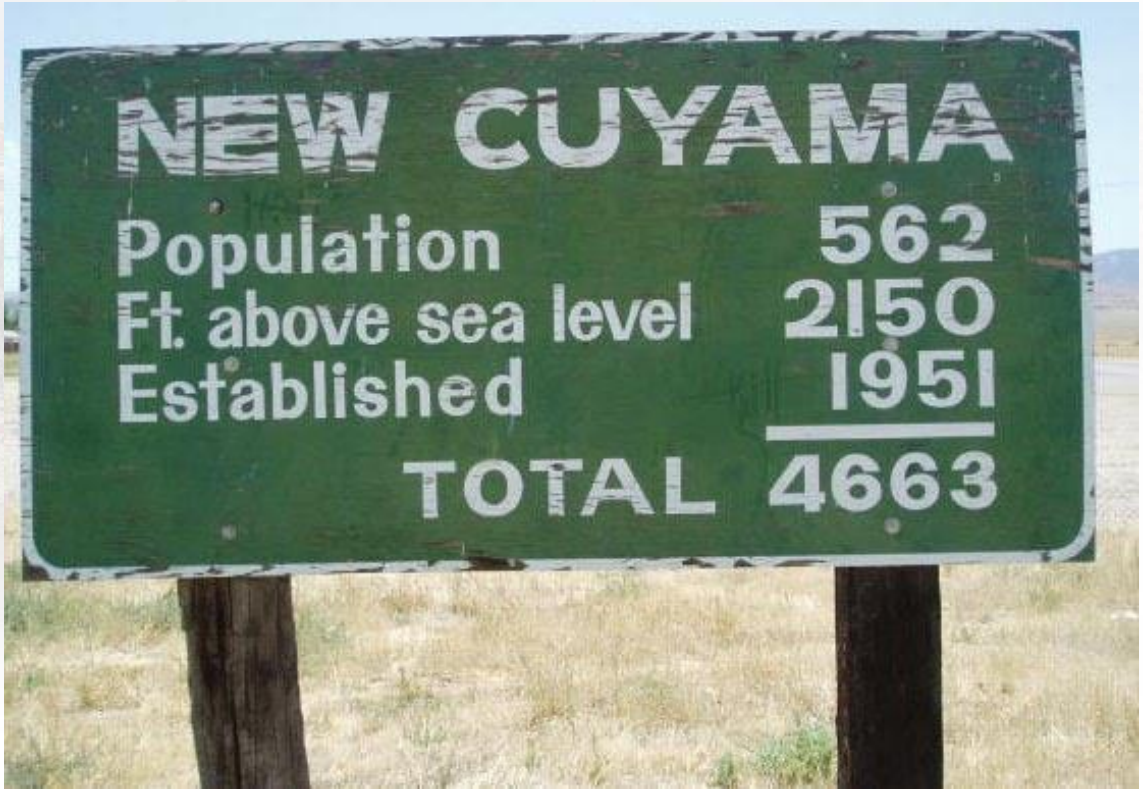
Combining multiple tables

Queries

BUDAPESTI MŰSZAKI
ÉS GAZDASÁGTUDOMÁNYI EGYETEM
Építőmérnöki Kar - építőmérnöki képzés 1782 óta

Fotogrammetria és Térinformatika Tanszék

Double check what to analyze

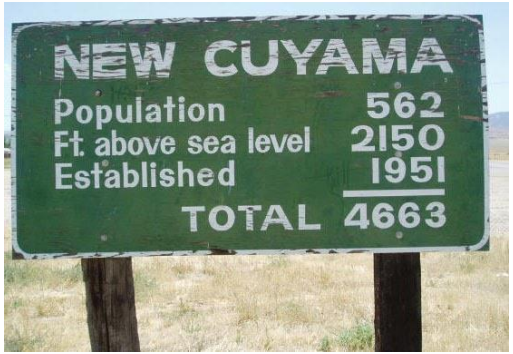


Combining tables

- Set operations
- Subqueries
- Cartesian product
- Natural join
- Theta join
  - *Inner*
  - *Left*
  - *Right*
  - *Full*

Requirements for set operations

- To perform set operations affecting two sets (R and S) followings have to be fulfilled
  - *R and S are agree on attributes*
  - *R and S are agree on the order of the attributes*



Set Operation I. - Union

Set items are listed below each other; no new column is created.

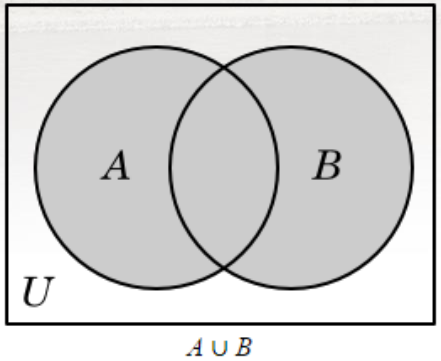
SQL: $SELECT * FROM A UNION SELECT * FROM B;$

name	grade
John Doe	2
Joe Average	3
Peter Common	5
Jane Doe	3

U

name	grade
Jane Smith	2
Bob Miggins	3

=



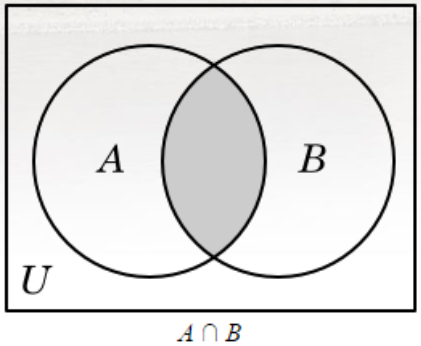
$A \cup B$

name	grade
John Doe	2
Joe Average	3
Peter Common	5
Jane Doe	3
Jane Smith	2
Bob Miggins	3

Set Operation II. - Intersect

Number of columns does not changes.

SQL: SELECT * FROM a INTERSECT SELECT * FROM b;



$A \cap B$

name	grade
John Doe	2
Joe Average	3
Peter Common	5
Jane Doe	3

$\cap$

name	grade
John Doe	2
Bob Miggins	3

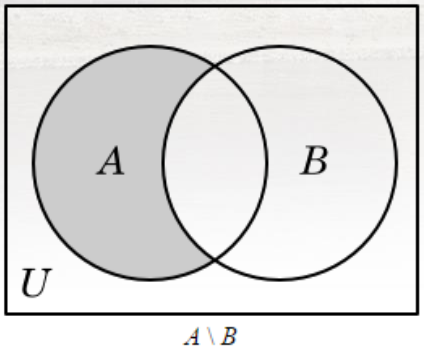
=

name	grade
John Doe	2

Set Operation II. - Except

Number of columns does not changes.

SQL: SELECT * FROM a EXCEPT SELECT * FROM b;



$A \setminus B$

name	grade
John Doe	2
Joe Average	3
Peter Common	5
Jane Doe	3

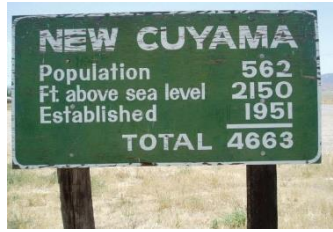
\

name	grade
John Doe	2
Bob Miggins	3

=

name	grade
Joe Average	3
Peter Common	5
Jane Doe	3

Subquery (Subselect)



We can combine two relations with subqueries. The number of columns is increasing because we want to put the data next to each other.
We usually avoid subqueries because they are slow.

movie(title, year, length, studioname, directorid)
director(id, name, address, income)

- `SELECT name FROM director WHERE id = (SELECT directorid FROM movie WHERE title = 'Lord of The Rings');`
- `SELECT * FROM movie WHERE directorid IN (SELECT id FROM director WHERE name LIKE 'P?ter*');`

Cartesian product

Creates a full combination without checking matching values.
Number of columns increasing.

SQL: `SELECT * FROM a,b;`

name	grade
John Doe	2
Joe Average	3

×

name	presence
John Doe	10
Joe Average	14
Bob Miggins	5

=

Result set is really big.

This is often combined with WHERE command to make it more useable.

A.name	grade	B.name	presence
John Doe	2	John Doe	10
John Doe	2	Joe Average	14
John Doe	2	Bob Miggins	5
Joe Average	3	John Doe	10
Joe Average	3	Joe Average	14
Joe Average	3	Bob Miggins	5

Natural Join

Combining two structurally different tables

- *both relations has a field name in common; this will be the basis of matching*
- *matched fields must agree on data type*
- *there is no other condition for matching in the query*
- *the result contains only one column of the matched attributes*

Natural Join

SQL: `SELECT * FROM r NATURAL JOIN s;`

name	grade
John Doe	2
Joe Average	3

⋈

name	presence
John Doe	5
Joe Average	14
Bob Miggins	11
Joe Average	12

=

name	grade	presence
John Doe	2	5
Joe Average	3	14
Joe Average	3	12

- Both source relations have “name” attribute
- There is only a single “name” column displayed in result

Dangling tuples

In a join operation the record exists in one of the relations; no matching pair is found in the other relation.

LEFT/RIGHT/FULL JOINs are suitable to include any type of dangling tuple.

Theta Join

- The connection does not need to be defined in the schema.
- Paired columns name might be different.
- At least one condition must be specified where each column of the two data sources is equal.
- Can also be used for subqueries.

In general, all theta results are true for:

- The columns used for matching from both relations appear in the result.
- If you find more than one pair, all combinations will be included in the result.

Theta Join - INNER

Based on cartesian product but includes matching condition.

SQL: `SELECT * FROM r INNER JOIN s ON r.attr1=s.attr2;`

name	grade
John Doe	2
Joe Average	3

$\bowtie_{A.\text{name} = B.fullname\ AND\ \text{presence} > 10}$

fullname	presence
John Doe	5
Joe Average	14
Bob Miggins	11
Joe Average	12

=

name	grade	fullname	presence
Joe Average	3	Joe Average	14
Joe Average	3	Joe Average	12

SELECT * FROM a INNER JOIN b ON a.name=b.fullname AND presence>10;

- Only rows existing in both relations are included (no dangling tuple).

Theta Join - LEFT

Based on cartesian product but includes matching condition.

SQL: `SELECT * FROM r LEFT JOIN s ON r.attr1=s.attr2;`

name	grade
John Doe	2
Joe Average	3

$\bowtie_{a.name = b.fullname\ AND\ presence > 10}$

fullname	presence
John Doe	5
Joe Average	14
Bob Miggins	11
Joe Average	12

=

name	grade	fullname	presence
John Doe	2	NULL	NULL
Joe Average	3	Joe Average	14
Joe Average	3	Joe Average	12

- All rows from left table are included. If there is no matching row, fields are filled with NULL values.

Theta Join - RIGHT

Based on cartesian product but includes matching condition.

SQL: `SELECT * FROM r RIGHT JOIN s ON r.attr1=s.attr2;`

name	grade
John Doe	2
Joe Average	3

$\bowtie_{\text{a.name} = \text{b.}fullname\ AND\ \text{presence} > 10}$

fullname	presence
John Doe	5
Joe Average	14
Bob Miggins	11
Joe Average	12

=

name	grade	fullname	presence
Joe Average	3	Joe Average	14
Joe Average	3	Joe Average	12
NULL	NULL	Bob Miggins	11

- All rows from left table are included. If there is no matching row, fields are filled with NULL values.
- John Doe is not included as there is an additional condition for presence.

Theta Join - FULL

Based on cartesian product but includes matching condition.

SQL: `SELECT * FROM r FULL OUTER JOIN s ON r.attr1=s.attr2…;`

name	grade
John Doe	2
Joe Average	3

$\bowtie_{\text{a.name} = \text{b.}fullname\ AND\ \text{presence} > 10}$

fullname	presence
John Doe	5
Joe Average	14
Bob Miggins	11
Joe Average	12

=

name	grade	fullname	presence
John Doe	2	NULL	NULL
Joe Average	3	Joe Average	14
Joe Average	3	Joe Average	12
NULL	NULL	Bob Miggins	11

Theta Join - FULL

```
SELECT * FROM a FULL OUTER JOIN b ON
a.name=b.fullname AND presence>10;
```

- Access has no FULL OUTER JOIN ☹
- Result includes dangling tuples from both table.
- John Doe is not included as there is an additional condition for presence.

Further commands

Queries

BUDAPESTI MŰSZAKI
ÉS GAZDASÁGTUDOMÁNYI EGYETEM
Építőmérnöki Kar - építőmérnöki képzés 1782 óta

Fotogrammetria és Térinformatika Tanszék

Sorting

SQL: `SELECT * FROM r ORDER BY attr1 ASC/DESC, attr2 ASC/DESC…;`

- `SELECT * FROM student ORDER BY place_of_birth DESC;`

neptun	name	place_of_birth
		Abádszalók
		Abádszalók
		Almásfüzitő
		Budapest
		…

neptun	name	place_of_birth
		Zalaegerszeg
		Zalaegerszeg
		Záhony
		Záhony
		…

Sorting based on multiple attributes

- Fields considered in the given order.
- `SELECT * FROM student ORDER BY name, grade;`

$\tau_{name,grade}($

name	grade	presence
John Doe	1	14
Joe Average	4	14
Jane Smith	2	14
Joe Average	3	10

$)=$

name	grade	presence
Jane Smith	2	14
Joe Average	**3**	**10**
Joe Average	**4**	**14**
John Doe	1	14

LIMIT NUMBER OF DISPLAYED ROWS

SQL: `SELECT TOP N * FROM r;`

- `SELECT TOP 1 name, grade FROM student ORDER BY grade;`

MySQL:
```
SELECT * FROM student ORDER BY grade
LIMIT 1;
```

If we look for exactly 1 row, then it worth to use LIMIT command as this speeds up the query (no further rows will checked).

Displaying only unique values

If there are repeated values, we can filter them.

SQL: `SELECT DISTINCT * FROM r;`

- `SELECT DISTINCT * FROM student;`

$\delta($

name	grade	presence
John Doe	1	14
Joe Average	3	14
John Doe	1	14
Jane Doe	3	10

$)=$

name	grade	presence
John Doe	1	14
Joe Average	3	14
Jane Doe	3	10

Grouping

Composing different groups based on one or multiple fields.

SQL: `SELECT attr1, attr2,… FROM r GROUP BY attr1, attr2,…;`

- `SELECT product FROM items GROUP BY product;`

$$\gamma_{product}(\pi_{product}($$

product	pcs
Bread	1
Croissant	2
Milk	1
Croissant	3

$$))$$

STEP-BY-STEP

$\pi_{product}($

product	pcs
Bread	1
Croissant	2
Milk	1
Croissant	3

$)=$

product
Bread
Croissant
Milk
Croissant

$\gamma_{product}($

product
Bread
Croissant
Milk
Croissant

$)=$

product
Bread
Croissant
Milk

Grouping + Aggregation

- As we have seen, grouping brings together the same elements on a given attribute.
- Other attributes also contain data, which can be aggregated by group in some way.
- Different functions can be used within the gamma operator for aggregation.
- These are: SUM, AVG, MIN, MAX, COUNT, FIRST, LAST

The structure of the result relation

- Divide the rows of the relation into groups. A group contains rows that have the same values for the grouping attributes in the list {attr1, attr2,…}. If there is no grouping attribute, the entire R relation forms a group.
- For each group, create a line item that contains:
  - *The grouping attributes of the group in question.*
  - *Summaries for the aggregation attributes of the list {attr1, attr2,…}.*

Grouping + Aggregation (Example - SUM)

- `SELECT product, SUM(pcs) FROM items GROUP BY product;`

$\gamma_{product, SUM(pcs)}($

product	pcs
Bread	1
Croissant	**3**
Milk	2
Croissant	**5**

$)=$

product	sumpcs
Bread	1
Croissant	**8**
Milk	2

FILTERING ON GROUP ATTRIBUTE

SQL: `SELECT attr1,AGGR(attr2)… FROM r GROUP BY attr1 HAVING AGGR(attr2) comp1 value1,…;`

- `SELECT AVG(grade) FROM student WHERE subject = 'Databases' AND grade > 3 GROUP BY year HAVING AVG(grade)>3.5;`

name	year	grade
John Doe	1990	4
Bob Miggins	1990	5
Jane Doe	1990	1
Jane Smith	1991	5
Peter Common	1991	4
Joe Average	1991	2
Alexander Tron	1992	4

SELECT year, AVG(grade) FROM student GROUP BY year;

year	grade
1990	3.33
1991	3.67
1992	4

name	year	grade
John Doe	1990	4
Bob Miggins	1990	5
Jane Doe	1990	1
Jane Smith	1991	5
Peter Common	1991	4
Joe Average	1991	2
Alexander Tron	1992	4

SELECT year, AVG(grade) FROM student GROUP BY year;

year	grade
1990	3.33
1991	3.67
1992	4

SELECT year, AVG(grade) FROM student WHERE grade>3 GROUP BY year;

Filtering of source rows

year	grade
1990	4.5
1991	4.5
1992	4

name	year	grade
John Doe		
Bob Miggins		
Jane Doe		
Jane Smith	1991	5
Peter Common	1991	4
Joe Average	1991	2
Alexander Tron	1992	4

SELECT year, AVG(grade) FROM student GROUP BY year;

year	grade
1990	3.33
1991	3.67
1992	4

SELECT year, AVG(grade) FROM student WHERE grade>3 GROUP BY year;
Filtering of source rows

year	grade
1990	4.5
1991	4.5
1992	4

SELECT year, AVG(grade) FROM student GROUP BY year HAVING AVG(grade)>=3.5;
Filtering of entire group

year	grade
1991	3.67
1992	4

Grouping + Aggregation (Example - For Multiple Attributes)

- `SELECT product, SUM(pcs), SUM(pcs*price) FROM items GROUP BY product;`

$$\gamma_{product,SUM(pcs),SUM(pcs*price)}($$

product	pcs	price
Bread	1	100
Croissant	2	100
Croissant	3	150
Bread	2	150
Milk	2	100

$$)=$$

product	sumpcs	sumpcsprice
Bread	3	400
Croissant	5	650
Milk	2	200

It is important that a column can only appear in the result relation if it is

- *the basis for grouping, or*
- *an aggregation operation is applied to it.*

Grouping + Aggregation (Example - COUNT)

- `SELECT product, COUNT(pcs) FROM items GROUP BY product;`

$$\gamma_{product,COUNT(pcs)}($$

product	pcs
Bread	1
Croissant	**3**
Milk	2
Croissant	**5**

$$)=$$

product	countpcs
Bread	1
Croissant	**2**
Milk	1

Grouping + Aggregation (Example - First)

- `SELECT product, FIRST(pcs) FROM items GROUP BY product;`

$$\gamma_{product, FIRST(pcs)}($$

product	pcs
Bread	1
Croissant	**3**
Milk	2
Croissant	5

$$)=$$

product	firstpcs
Bread	1
Croissant	**3**
Milk	2

Alias (renaming)

In some cases, we want to rename the columns in the query:

- Columns generated by functions or formulas
- Avoid field name conflicts when joining

SQL: … AS…

- `SELECT AVG(grade) AS average, presence/14*100 AS presence_percent FROM student;`

Moreover, the AS command can be omitted:

- `SELECT AVG(grade) average, presence/14*100 presence_percent FROM student;`

Moreover, for sub-queries, we can even name the query units so that it can be referenced:

- `SELECT c.currency, c.ratetohuf, c.date FROM currencyrate AS c LEFT JOIN (SELECT currency, MAX(ratetohuf) AS max FROM currencyrate GROUP BY currency) AS m ON c.currency=m.currency AND c.ratetohuf=m.max;`

Alias

- `SELECT name AS fullname FROM student;`

$\pi_{name \rightarrow fullname}($

name	grade	presence
John Doe	1	14
Joe Average	3	14
John Doe	1	14
Jane Doe	3	10

$) =$

fullname
John Doe
Joe Average
John Doe
Jane Doe

- `SELECT name AS fullname FROM student WHERE fullname='John Doe';`

$\sigma_{fullname='John\ Doe'}(\pi_{name \rightarrow fullname}($

name	grade	presence
John Doe	1	14
Joe Average	3	14
Bob Miggins	1	14
Jane Doe	3	10

$)) =$

fullname
John Doe

Order of commands

This is a strict order:

```
SELECT
DISTINCT
TOP
FROM
JOIN      ON
WHERE
GROUP BY
HAVING
ORDER BY
```

Syntax

- In the case of selection (filtering) and theta join, if several attributes are queried, separate them with AND / OR! However, it is enough to separate the column names to be displayed by the projection with a comma!
- If you are filtering on a text value or theta join, enclose the text value in quotation marks.
- If there are more than one relation in the query, the name of the containing relation (relation.fieldname) should appear before the field names.
- If the names of the columns on which the connection is based have different names for the join, the theta join must be used.
- In the case of a theta join, the conditions must include the equality of the two fields on which the connection is based.
- Dates are written as text, so they are enclosed in quotation marks, but if specified in a standard format, Database Management Systems can interpret them numerically.

Cheat Sheet

SELECT attribute(s) FROM table(s) WHERE property(s);

- `SELECT name FROM student WHERE name = 'John Doe';`

Attributes & tables are separated by comas
Conditions are combined by logical operators: AND, OR, XOR, NOT
Comparison operators: =, <>, >, <, =>, <=, LIKE, BETWEEN, IN, IS
DISTINCT
aliases (AS)
Special characters: Access-ben: *, ?, # (Some DBMS: *, _, %)
Escape characters (\)
Quotations: ', "
Lower and uppercase (normally case insensitive)

Query examples

Exercise

BUDAPESTI MŰSZAKI
ÉS GAZDASÁGTUDOMÁNYI EGYETEM
Építőmérnöki Kar - építőmérnöki képzés 1782 óta

Fotogrammetria és Térinformatika Tanszék

EXAMPLE 1

Using SQL, enter the names of the students who passed the subject.

S		
name	**grade**	**presence**
John Doe	3	8
Bob Miggins	2	14
Jane Doe	5	10
Joe Average	1	14

$$SELECT\ name\ FROM\ S\ WHERE\ grade > 1\ AND\ presence > 10;$$

Example 2

lecturers	
lecturer	**department_code**
Zoltán Koppányi	FMT
Tamás Lovas	FMT
Tamás Tuchband	AGT
John Doe	NULL

departments	
department_code	**department_name**
FMT	Fotogrammety
AGT	Surveying
OCT	Organic chemistry

Example 2 – Natural Join

```
SELECT * FROM lecturers NATURAL JOIN
departments;
```

lecturer	department_code	department_name
Zoltán Koppányi	FMT	Fotogrammetry
Tamás Lovas	FMT	Fotogrammetry
Tamás Tuchband	AGT	Surveying

EXAMPLE 2 - LEFT JOIN

```
SELECT * FROM lecturers LEFT JOIN
departments ON lecturers.department_code =
departments.department_code;
```

lecturers.lecturer	lecturers.department_code	departments.department_code	departments.department_name
Zoltán Koppányi	FMT	FMT	Fotogrammetry
Tamás Lovas	FMT	FMT	Fotogrammetry
Tamás Tuchband	AGT	AGT	Surveying
John Doe	NULL	NULL	NULL

EXAMPLE 2 - RIGHT JOIN

```
SELECT * FROM lecturers RIGHT JOIN
departments ON lecturers.department_code =
departments.department_code;
```

lecturers.lecturer	lecturers.department_code	departments.department_code	departments.department_name
Zoltán Koppányi	FMT	FMT	Fotogrammetry
Tamás Lovas	FMT	FMT	Fotogrammetry
Tamás Tuchband	AGT	AGT	Surveying
NULL	NULL	OCT	Organic chemistry

Exercise – schema diagram

A			
name	**subject**	**point**	**presence**
John Doe	Math	50	8
John Doe	Drawing	60	14
Peter Common	Statics	45	10
Joe Average	Math	15	14

B	
name	**year**
John Doe	1
Bob Miggins	2
Peter Common	1
Joe Average	1

C	
class	**minpoint**
Math	40
Drawing	60
Statics	50

Exercise - questions

1) List students enrolled to drawing!

2) List rookie (first year) students.

3) List subjects where minimum points over 45!

4) List math students and their year!

5) List all passed students with subject names (without checking attendance)!

6) List all passed rookie students with subject names (without checking attendance)!

EXERCISE - SOLUTIONS

1) `SELECT name FROM A WHERE subject='Drawing';`
2) `SELECT name FROM B WHERE year=1;`
3) `SELECT class FROM C WHERE minpoint>45;`
4) `SELECT name, year FROM A NATURAL JOIN B WHERE A.subject='Math';`
5) `SELECT name, subject FROM A INNER JOIN C ON A.subject=C.class AND A.point>C.minpoint;`
6) `SELECT name, subject FROM A NATURAL JOIN B INNER JOIN C ON A.subject=C.class AND A.point>C.minpoint WHERE year=1;`

Data definition

BUDAPESTI MŰSZAKI
ÉS GAZDASÁGTUDOMÁNYI EGYETEM
Építőmérnöki Kar - építőmérnöki képzés 1782 óta

Fotogrammetria és Térinformatika Tanszék

Creating database

- SQL: `CREATE DATABASE databasename;`
- permissions
  - *user*
  - *connecting from*
- encoding (UTF8, LATIN2 [ISO-8859-2])
- templates (e.g. for spatial database)
- Switching to database
  - *Explicit during connection*
  - `USE databasename;`
- Allowed characters: [_a-zA-Z0-9] (cannot start with number!)

Hint:
Relational database schema

Relational schema: student(neptun: String, name: String, date_of_birth: Integer)

Relational database schema:

student
neptun: String name: String date_of_birth: Integer ...

Creating table

```
CREATE TABLE student
(
      neptun VARCHAR(6),
      name VARCHAR(50),
      date_of_birth INTEGER,
      grade DOUBLE PRECISION,
      PRIMARY KEY (neptun)
);
```

Creating table

```
CREATE TABLE tablaname
```
…:

- *DEFAULT*
- *NULL /NOT NULL*
- *UNIQUE* *(!= PRIMARY KEY!)*
- *AutoNumber/AUTO INCREMENT/Sequence*

Modifying table

SQL: `ALTER TABLE tablaname ADD columname type;`
- *`ALTER TABLE student ADD place_of_birth VARCHAR(50);`*

SQL: `ALTER TABLE tablaname DROP columname;`
- *`ALTER TABLE student DROP name;`*

SQL: `ALTER TABLE tablaname RENAME TO new_tablaname;`
- *`ALTER TABLE student RENAME TO bme_student;`*

SQL: `DROP tablaname;`
- *`DROP student;`*

SQL: `TRUNCATE tablaname;`
- *`TRUNCATE student;`*

In case of modifying, please do not forget to adjust joined tables as well!

Data manipulation

BUDAPESTI MŰSZAKI
ÉS GAZDASÁGTUDOMÁNYI EGYETEM
Építőmérnöki Kar - építőmérnöki képzés 1782 óta

Fotogrammetria és Térinformatika Tanszék

Upload data

Insertion

- *SQL:* `INSERT INTO tablaname [(attribute1, attribute2, …)] VALUES (value1, value2, …);`
- `INSERT INTO student (neptun, name, year, grade) VALUES ('ABCDEF', 'John Doe', 1993, 4.5);`
- *SQL:* `LOAD DATA`
- *SQL:* `COPY`

Data modification

Don't forget to use selection; you'll modify/delete all records otherwise.

Update

- *SQL:* `UPDATE tablaname SET attribute = newvalue WHERE condition;`
- `UPDATE student SET grade = 4.6 WHERE name = 'John Doe';`

Deletion

- *SQL:* `DELETE FROM tablaname WHERE attribute = value;`
- `DELETE FROM student WHERE name = 'John Doe';`

Conclusion

- Fundamentals
- Analytic operations
- Data definition
- Data manipulation

Thank you for your attention!

Questions?

BUDAPESTI MŰSZAKI
ÉS GAZDASÁGTUDOMÁNYI EGYETEM
Építőmérnöki Kar - építőmérnöki képzés 1782 óta

Fotogrammetria és Térinformatika Tanszék