Database systems I

Radim Bača

radim.baca@vsb.cz

dbedu.cs.vsb.cz

VSB TECHNICAL UNIVERSITY OF OSTRAVA | FACULTY OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE | DEPARTMENT OF COMPUTER SCIENCE

EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

Content

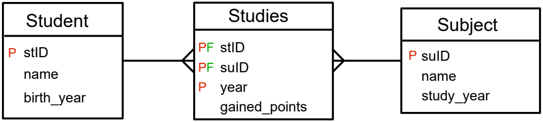
- Aggregation
- 3 value logic
- Task specification

Aggregate functions

- Aggregate functions and SELECT:

  SELECT $A_1, \dots, A_n$ ← we can use functions such as AVG, MIN, MAX, COUNT, SUM
  FROM R_1
  JOIN R_n ON *join_condition*
  WHERE *condition*

Example: Simple queries with aggregate functions



Student

P stID

name

birth_year

Studies

PF stID

PF suID

P year

gained_points

Subject

P suID

name

study_year

- *Find an average birth year of all students.*

  ```
  SELECT AVG(birth_year) FROM Student
  ```

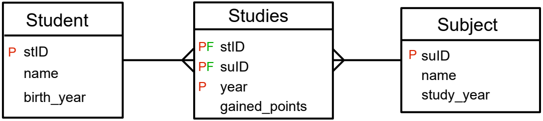
- *Find a birth year of the oldest student.*

  ```
  SELECT MIN(birth_year) FROM Student
  ```

- *Find number of all students.*

  ```
  SELECT COUNT(*) FROM Student
  ```

Example: Aggregate functions and NULL values



Student
P stID
name
birth_year

Studies
PF stID
PF suID
P year
gained_points

Subject
P suID
name
study_year

- *Find number of all students.*
  - `SELECT COUNT(*) FROM Student`
  - `SELECT COUNT(birth_year) FROM Student`
  - Aggregate functions ignore NULL values
  - Thus, the first above query is correct since the second one answers the question: *For how many students do we know the birth year?*

GROUP BY clause

- SELECT with grouping:

  SELECT $A_1, \dots, A_n$
  FROM R_1
  JOIN R_n ON *join_condition*
  WHERE *condition*
  GROUP BY $A_1, \dots, A_n$

GROUP BY clause

- SELECT with grouping:

  SELECT $A_1, \ldots, A_n$
  FROM R_1
  JOIN R_n ON *join_condition*
  WHERE *condition*
  GROUP BY $A_1, \ldots, A_n$ - defines attributes that specify groups in the data

GROUP BY Operation

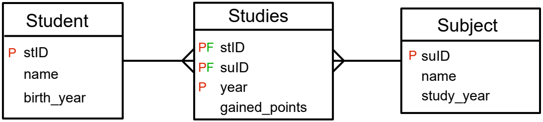
Studies

stID	suID	year	points
1	35	2010	23
1	35	2011	55
1	21	2010	89
2	11	2011	98
2	21	2011	null
3	46	2011	null

- GROUP BY create logical groups in the input relation

-
  ```
  SELECT stID, COUNT(*)
  FROM Studies
  GROUP BY stID
  ```

Example: GROUP BY with more attributes



- *Find number of students of particular subjects in particular years.*

```
SELECT P.suID, S.year, COUNT(*)
FROM Subject P
JOIN Studies S ON P.suID=S.suID
GROUP BY P.suID, S.year
```

- How to include students who did not studied anything?

GROUP BY Rule

Behind the SELECT we can have outside aggregation function only attributes that are behind GROUP BY!

```
SELECT year, COUNT(*)
  FROM Subject
  GROUP BY year
```

GROUP BY Rule

- *Find number of students of particular subjects in particular years.*

  ```
  SELECT P.name, S.year, COUNT(*)
  FROM Subject P
  JOIN Studies S ON P.suID=S.suID
  GROUP BY P.suID, S.year
  ```

- MS SQL Server and Oracle will not perform this query - they both require that GROUP BY is followed by all attributes following the SELECT clause except those attributes following SELECT which appear only in aggregate functions after SELECT (these can but do not have to follow GROUP BY)
- Therefore, even in this case when `name` is uniquely determined by the `suID` value, the query will not be performed

GROUP BY Rule

- *Find number of students of particular subjects in particular years.*

  ```
  SELECT P.name, S.year, COUNT(*)
  FROM Subject P
  JOIN Studies S ON P.suID=S.suID
  GROUP BY P.suID, S.year
  ```

- MS SQL Server and Oracle will not perform this query - they both require that GROUP BY is followed by all attributes following the SELECT clause except those attributes following SELECT which appear only in aggregate functions after SELECT (these can but do not have to follow GROUP BY)
- Therefore, even in this case when `name` is uniquely determined by the `suID` value, the query will not be performed

GROUP BY Rule

- Some database systems such as MYSQL perform also this query:

  ```
  SELECT suID, year, COUNT(*)
  FROM Subject
  GROUP BY year
  ```

- Note that more subjects with different suIDs can correspond to a single row - suID in the result is then random!
- Personally, I consider this behavior to be confusing, when less experienced users are not aware of the fact that a query returns random values

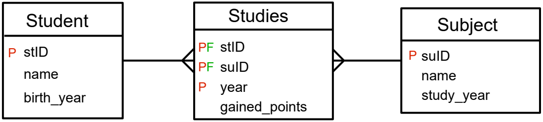
GROUP BY Rule

- Some database systems such as MYSQL perform also this query:

```
SELECT suID, year, COUNT(*)
FROM Subject
GROUP BY year
```

- Note that more subjects with different suIDs can correspond to a single row - suID in the result is then random!
- Personally, I consider this behavior to be confusing, when less experienced users are not aware of the fact that a query returns random values

Example: GROUP BY, LEFT JOIN and cond.



Student

P stID

name

birth_year

Studies

PF stID

PF suID

P year

gained_points

Subject

P suID

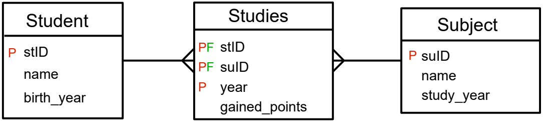
name

study_year

- *List names of all subjects together with numbers of students who study or studied particular subjects in year 2010.*

  -
    ```
    SELECT Su.name, COUNT(distinct Su.suID)
    FROM Subject Su
    LEFT JOIN Studies Sy ON Su.suID = Sy.suID
    WHERE Sy.year = 2010
    GROUP BY Su.name
    ```

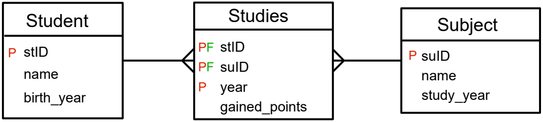
Example: GROUP BY, LEFT JOIN and cond.



- *List names of all subjects together with numbers of students who study or studied particular subjects in year 2010.*

  -
    ```
    SELECT Su.name, COUNT(distinct Su.suID)
    FROM Subject Su
    LEFT JOIN Studies Sy ON Su.suID = Sy.suID
    WHERE Sy.year = 2010
    GROUP BY Su.name
    ```
  - Incorrect! Condition has to be part of left join.

Example: GROUP BY, LEFT JOIN and cond.



- *List names of all subjects together with numbers of students who study or studied particular subjects in year 2010.*

  -
    ```
    SELECT Su.name, COUNT(distinct Su.suID)
    FROM Subject Su
    LEFT JOIN Studies Sy ON Su.suID = Sy.suID
         and Sy.year = 2010
    GROUP BY Su.name
    ```

Subquery Behind SELECT

- We can rewrite the GROUP BY queries that there is no GROUP BY clause

```
SELECT year, COUNT(*)
FROM Subject
GROUP BY year
```

$\Rightarrow$

Subquery Behind SELECT

- We can rewrite the GROUP BY queries that there is no GROUP BY clause
- The solution uses *dependent subqueries* behind SELECT

```
SELECT year, COUNT(*)
FROM Subject
GROUP BY year
```

⇒

```
SELECT distinct year, (
   SELECT COUNT(*)
   FROM Subject s2
   WHERE s1.year = s2.year
  )
FROM Subject s1
```

Subquery Behind SELECT

- In certain situations it may be easier to use dependent subqueries

```
SELECT st.stID,
       COUNT(*)
FROM Student st
LEFT JOIN Studies ss
  ON st.stID = ss.stID
GROUP BY st.stID
```

⇒

Subquery Behind SELECT

- In certain situations it may be easier to use dependent subqueries
- We can avoid LEFT JOIN as well [1]

```
SELECT st.stID,
       COUNT(*)
FROM Student st
LEFT JOIN Studies ss
  ON st.stID = ss.stID
GROUP BY st.stID
```

⇒

```
SELECT st.stID, (
    SELECT COUNT(*)
    FROM Studies ss
    WHERE st.stID = ss.stID
  )
FROM Student st
```

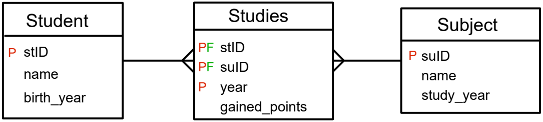
[1] Curious which solution is better? https://github.com/RadimBaca/SQLart/tree/master/articles/Postgre_aggregation

Having

- SELECT with grouping:

  SELECT $A_1, \ldots, A_n$
  FROM R_1
  JOIN R_2 ON *join_condition*
  WHERE *condition*1
  GROUP BY $A_1, \ldots, A_n$
  HAVING *condition*2 - condition using an aggregation

Example: HAVING

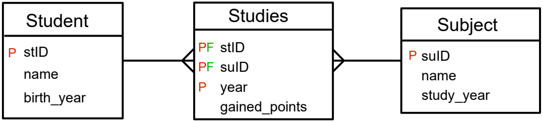


- *List names of all subjects that are/were studied at least by two students.*

```
SELECT suID FROM Studies
GROUP BY suID HAVING Count(distinct stID) > 1
```

- The HAVING clause primarily simplifies notation of such queries
- Again we can use dependent subqueries to avoid HAVING clause as well

Example: HAVING

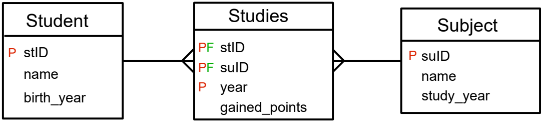


- *List names of all subjects that are/were studied at least by two students.*

```
SELECT suID FROM Studies
GROUP BY suID HAVING Count(distinct stID) > 1
```

- The HAVING clause primarily simplifies notation of such queries
- Again we can use dependent subqueries to avoid HAVING clause as well

Example: HAVING



- *List names of all subjects that are/were studied at least by two students.*

```
SELECT distinct s1.suID
FROM Studies s1
WHERE 1 < (SELECT Count(distinct stID)
           FROM Studies s2
           WHERE s1.suID = s2.suID)
```

Clause Priority

1. `FROM`
2. `JOIN`
3. `WHERE`
4. `GROUP BY`
5. `HAVING`
6. `SELECT`
7. `DISTINCT`
8. `ORDER BY`
9. `FETCH/NEXT`

- Semantic order
- Not a query processing order!

Clause Priority

1. `FROM`
2. `JOIN`
3. `WHERE`
4. `GROUP BY`
5. `HAVING`
6. `SELECT`
7. `DISTINCT`
8. `ORDER BY`
9. `FETCH/NEXT`

- Semantic order
- Not a query processing order!
- However this *is not* possible due to this ordering:

-
  ```
  SELECT count(*) c
  FROM Student
  GROUP BY birth_year
  HAVING c = 1
  ```

Order of Evaluation

1. FROM
2. JOIN
3. WHERE
4. GROUP BY
5. HAVING
6. SELECT
7. DISTINCT
8. ORDER BY
9. FETCH/NEXT

- Semantic order
- Not a query processing order!
- And this *is* possible:

-
  ```
  SELECT count(*) c
  FROM Student
  GROUP BY birth_year
  ORDER BY c
  ```

NULL

- `Null` value represent missing data
- We define whether an attribute can have a `null` value or not when we create a table (lecture 6)
- Primary key are the only type of attributes that can not have `null` values at all

Three Value Logic (3VL)

- Every condition in SQL is evaluated into `true`, `false` or `unknown`
- Row is in the result only if the result of the condition is `true`
- Examples when a predicate returns unknown:
  - `attribute = null`
  - `unknown and true`
  - `unknown or false`

3VL Paradoxes

- `P or not P` is not always true
-
  ```
  SELECT *
  FROM student
  WHERE birth_year = NULL and
      NOT birth_year = NULL
  ```
- The predicate is evaluated to `unknown` for each row since the comparisons are evaluated to `unknown`

3VL Paradoxes

- `P or not P` is not always true

-
  ```
  SELECT *
  FROM student
  WHERE birth_year = NULL and
      NOT birth_year = NULL
  ```
- The predicate is evaluated to `unknown` for each row since the comparisons are evaluated to `unknown`

3VL Paradoxes

- `Attribute NOT IN (null)` is never true

-
  ```
  SELECT *
  FROM student
  WHERE birth_year NOT IN (1980, null)
  ```

⇓

-
  ```
  SELECT *
  FROM student
  WHERE birth_year != 1980 and
        birth_year != null
  ```

3VL Paradoxes

- `Attribute NOT IN (null)` is never true
-
  ```
  SELECT *
  FROM student
  WHERE birth_year NOT IN (1980, null)
  ```

⇓

-
  ```
  SELECT *
  FROM student
  WHERE birth_year != 1980 and
        birth_year != null
  ```

Task

- Task formulate the requirements
- Features:
  - Description in a text form
  - Ambiguous, incomplete, inaccurate
  - Understandable to both the client and the developer

Task Specification

Task specification constructively answers the following questions:

- WHY? The motivation behind the IS creation
- FOR WHAT? Problems solved by the IS
- WHO? Roles of users that will use the IS
- INPUTS? The entities that we want to store in our system
- OUTPUTS? The most important views
- FUNCTIONS? Non trivial functions
- HISTORY

Task Specification

Task specification constructively answers the following questions:

- WHY? The motivation behind the IS creation
- FOR WHAT? Problems solved by the IS
- WHO? Roles of users that will use the IS
- INPUTS? The entities that we want to store in our system
- OUTPUTS? The most important views
- FUNCTIONS? Non trivial functions
- HISTORY

WHO?

- Describe roles of users in the company that will be using your IS
- Those people have specific job/role in the company, describe it
- Do not write roles USER and ADMIN! ADMIN is not a role in a company!

Example - WHY?

- We need a room reservation system in our building
- There is no central authority and we often argue about the rooms between the colleagues
- Some colleagues even take personally if the room is not empty for their meeting

Example - FOR WHAT?

- The software will provide a simple way to organize reservations in a building
- The system will not only provide a way to reserve a meeting room, but it will also offer a possibility to define a single authority for a meeting room
- The authority will have an authorization to cancel a reservation in that room and the software will implement a notification system as well that will send an email in such case

Example - WHO?

- Company employee – a person that work in a company and have to organize a meeting time to time. This is the person that will have a possibility to create a reservation.
- Company manager – a person that manage a larger group of people and need a meeting room for them and for himself quite often.
- Codebook keeper – a person that will keep the codebooks in the database up-to-date

Example - INPUTS?

- **Person** information such as name, email, office, phone and flag indicating whether it is a office manager or not (Company employee, Company manager).
- For each **room** we store its building, floor, room number and room authority (Codebook keeper).
- Information about a **reservation** such as reservation date, meeting interval, room and person who created the reservation (Company employee, Company manager).
- Each person can create many reservations.

Example - OUTPUTS?

- List of an employee future reservations that are sorted according to the meeting time ascending order. Employee may also list the history of his reservations according to a specified time interval. (Company employee)
- List of a future room reservations. The system should offer also a week schedule for a room. (Company employee, Company manager)
- List of free rooms for a specific time. (Company employee)

Example - FUNCTIONS?

- Possibility to create a reservation in a meeting room even though there are already reservations (only for a company manager). In such a case, the system will propose an empty meeting room in a similar time for each overlapping reservation.
- The system will notify by email each user whose reservation was shifted.

Example - HISTORY

- If a reservation is moved by somebody else than the owner, then we need to keep the original reservation in the system. In other words, we need to track back the history of reservation changes made by a company manager.

The Project Topic

When selecting the topic of the project consider the following:

- The topic have to be familiar to you
    - Do not select bank accounts or bank loans if you never worked in a bank
    - Do not select food delivery if you have no experience with it
- It should not be easily replaceable by a content management system
    - Database of battles and persons in a World War II
    - Database of space objects

References

- Course home pages `http://dbedu.cs.vsb.cz`