**Institut für Informatik**
**Datenbanken und Informationssysteme**
Prof. Dr. Stefan Conrad
Thomas Germer

WiSe 24/25
Assignment 1
Due October 15, 8:00 AM

# Relational Databases and Data Analysis

- Please read Assignment 0 if you have not done so yet.
- For this assignment, you can hand in your solution as PDF or as scans of handwritten solutions converted to PDF. The file must be named `solution-1.pdf`. Use `zipme.py` to verify that you named the file correctly and upload the resulting `tar.gz` archive in ILIAS.

## Exercise 1 *Relational Model* (2 + 3 Points)

**(a)** You are given a relation $R(A, B, C, D)$. Identify all sets of attributes which could be keys.

| A | B | C | D |
|---|---|---|---|
| 1 | 1 | 2 | 1 |
| 1 | 3 | 4 | 1 |
| 2 | 1 | 4 | 4 |
| 2 | 3 | 2 | 1 |
| 2 | 4 | 3 | 2 |
| 3 | 2 | 1 | 2 |
| 3 | 4 | 2 | 1 |
| 4 | 1 | 3 | 4 |
| 4 | 3 | 3 | 2 |
| 4 | 4 | 1 | 4 |

- What is a key?
  - A minimal set of attributes, the values of which can uniquely identify a tuple within a table.
- All rows are unique, so the set of attributes $\{A, B, C, D\}$ uniquely identifies each row (tuple).
- But is $\{A, B, C, D\}$ minimal, i.e. is a true subset also uniquely identifying?
  - No, because $\{A, B\}$ also uniquely identifies each row.
- Which subsets of $\{A, B, C, D\}$ uniquely identify each row?
  - Attributes $\{A, B, C, D\}$ uniquely identify each row
  - Attributes $\{B, C, D\}$ uniquely identify each row
  - Attributes $\{A, C, D\}$ uniquely identify each row
  - Attributes $\{A, B, D\}$ uniquely identify each row
  - Attributes $\{A, B, C\}$ uniquely identify each row
  - Attributes $\{C, D\}$ have common value: $(2, 1)$
  - Attributes $\{B, D\}$ have common value: $(3, 1)$
  - Attributes $\{B, C\}$ uniquely identify each row

  - Attributes $\{A, D\}$ have common value: $(1, 1)$
  - Attributes $\{A, C\}$ have common value: $(4, 3)$
  - Attributes $\{A, B\}$ uniquely identify each row
  - Attributes $\{D\}$ have common value: $(1)$
  - Attributes $\{C\}$ have common value: $(4)$
  - Attributes $\{B\}$ have common value: $(1)$
  - Attributes $\{A\}$ have common value: $(1)$
  - Attributes $\{\}$ have common values: $(1)$

- Is $\{A, B\}$ minimal?
  - Yes! Neither $\{A\}$ nor $\{B\}$ can uniquely identify each row, because the domain of each row contains 4 values, but we have 10 rows.
- Could $\{A\}$, $\{B\}$, $\{C\}$ or $\{D\}$ have been minimal? Did we have to check them?
  - No! Due to the pigeonhole principle, there will be at least 7 rows which can not be identified.
  - Pigeonhole principle: Given 10 pigeons and 9 pigeonholes, at least one pigeonhole must contain more than one pigeon.
  - Optimal, i.e. 7 identifiable rows in terms of pigeons: Three pigeons per pigeonhole, seven unhappy pigeons in fourth pigeonhole.
  - Can not be less than 7 rows: Moving pigeons from fourth pigeonhole would make other pigeons unhappy.
  - Could be more than 7 rows: Two pigeons each in first two pigeonholes, three pigeons each in other two pigeonholes.
  - Funny application of pigeonhole principle: There are at least two people in London with the same number of hairs on their head.
    - Counting all hairs of all people would be a lot of work.
    - Reasonable assumptions:
    - Average number of hairs: 150,000
    - Maximum number of hairs: 1,000,000
    - $\rightarrow$ 850,000 possible hair counts
    - Population of London mid 2019: 8.961.989 [1]
- Also minimal: $\{B, C\}$
- $\{B, C, D\}$ uniquely identifying, but not minimal, because $\{B, C\}$ is subset
- $\{A, C, D\}$ minimal (no subset can uniquely identify all attributes)
- $\{A, B, D\}, \{A, B, C\}$ not minimal, because $\{A, B\}$ is subset
- What about $\{\}$?
  - Without values, rows can not be identified. Exception: Database with single row.
- Summarizing, the keys are: $\{A, B\}, \{B, C\}, \{A, C, D\}$

[1] `https://www.ons.gov.uk/peoplepopulationandcommunity/populationandmigration/populationestimates/datasets/populationestimatesforukenglandandwalesscotlandandnorthernireland`

**(b)** Assuming the following attribute names with the same values as before, which keys would still be a reasonable choice for a production database given the new domain knowledge? Consider that more rows could be added in the future. The attribute `user_order_id` can be assumed to be unique among orders by the *same* user.

**Orders**

| `user_id` | `user_order_id` | `amount` | `product_id` |
|---|---|---|---|
| 1 | 1 | 2 | 1 |
| 1 | 3 | 4 | 1 |
| 2 | 1 | 4 | 4 |
| 2 | 3 | 2 | 1 |
| 2 | 4 | 3 | 2 |
| 3 | 2 | 1 | 2 |
| 3 | 4 | 2 | 1 |
| 4 | 1 | 3 | 4 |
| 4 | 3 | 3 | 2 |
| 4 | 4 | 1 | 4 |

- Keys from before: $\{A, B\}, \{B, C\}, \{A, C, D\}$
- {`user_id`, `user_order_id`}
  - Reasonable choice. `user_order_id` is specific to user and `user_id` should be unique, so the combination should be unique for each order.
- {`user_order_id`, `amount`}
  - Not a good choice. Two different users could order the same amount and happen to have the same `user_order_id`. The thusly created row would not be identifiable by this key.
- {`user_id`, `amount`, `product_id`}
  - Not a good choice. The same user could order the same amount of the same product again.
- Other bad ideas for keys:
  - Addresses (multiple people might live at the same address)
  - Names (multiple people might have the same name)
  - Name + date of birth (still not unique, see Birthday Problem)
  - Email addresses (might change, for example if email provider shuts down or user stops paying services)
  - Usernames (might have to change, for example due to privacy laws or indecent names)

**Exercise 2** *Relational Algebra* (5 Points)

In these exercises, we will follow the notation by Heuer, Sattler and Saake from the book “Datenbanken – Konzepte und Sprachen”, of which various versions are freely available online in the library. You might have to log into the university network to gain access.

For your convenience, the most important operators are listed below. Also have a look at the list of common mistakes on the last page.

The university of Innsbruck also provides a nice online tool to play around with relational algebra.

**Projection**
Definition:

$$\pi_X(r) = \{t(X) \mid t \in r\}$$

Example: Restrict `Orders` to column `user_id`

$$\pi_{\text{user\_id}}(\text{Orders})$$

| `user_id` |
|---|
| 1 |
| 2 |
| 3 |
| 4 |

**Selection**
Definition:

$$\sigma_F(r) = \{t \mid t \in r \wedge F(t) = \textbf{true}\}$$

Example: Select user IDs with value less than 2

$$\sigma_{\text{user\_id} \, < \, 2}(\pi_{\text{user\_id}}(\text{Orders}))$$

| `user_id` |
|---|
| 1 |

**Rename**
Definition:

$$\rho_{\text{B}\leftarrow\text{A}}(r) = \{t' \mid \exists t \in r : t'(R - A) = t(R - A) \wedge t'(B) = t(A)\}$$

Example: Rename `user_id` to `id`

$$\rho_{\text{id}\leftarrow\text{user\_id}}(\sigma_{\text{user\_id}=1}(\pi_{\text{user\_id}}(\text{Orders})))$$

| `id` |
|---|
| 1 |

**(Natural) Join**
Definition:

$$r_1 \bowtie r_2 = \{t(R_1 \cup R_2) \mid \forall i \in \{1,2\}\, \exists t_i \in r_i : t_i = t(R_i)\}$$

**WARNING:** This operator joins over **all** columns with the same name!

Example: Join all `Orders` with user IDs with value 1.

$$\text{Orders} \bowtie \sigma_{\text{user\_id}=1}(\pi_{\text{user\_id}}(\text{Orders}))$$

| user_id | user_order_id | amount | product_id |
|---|---|---|---|
| 1 | 1 | 2 | 1 |
| 1 | 3 | 4 | 1 |

**Miscellaneous operators**

- Union: $r_1 \cup r_2 = \{t \mid t \in r_1 \vee t \in r_2\}$
- Difference: $r_1 - r_2 = \{t \mid t \in r_1 \wedge t \notin r_2\}$
- Intersection: $r_1 \cap r_2 = \{t \mid t \in r_1 \wedge t \in r_2\} = r_1 - (r_1 - r_2)$
- Division (assuming $r_1(R_1), r_2(R_2), R_2 \subset R_1, R' = R_1 - R_2$):

  $r_1 \div r_2 = \pi_{R'}(r_1) - \pi_{R'}((\pi_{R'}(r_1) \bowtie r_2) - r_1)$

  $r'(R') = \{t \mid \forall t_2 \in r_2\, \exists t_1 \in r_1 : t_1(R') = t \wedge t_1(R_2) = t_2\}$

Given the following relational model of a supermarket chain where underlined attributes indicate primary keys and overlined attributes indicate foreign keys:

`product(id, price, name)`

`store(id, city)`

`customer(id, firstname, lastname)`

`sold_in(product_id, store_id)` with foreign keys `product_id` referencing the attribute `id` of `product` and `store_id` referencing the `id` of `store`

`order(id, customer_id, product_id, amount)` with foreign keys `customer_id` referencing `id` of `customer` and `product_id` referencing `id` of `product`

Formulate the following queries using relational algebra. In this lecture, only the following operators are allowed: $\pi, \sigma, \bowtie, \rho(\equiv \beta), \leftarrow, \wedge, \vee, =, \leq, \geq, \times, \neq, \neg, -, \div$

**(a)** Find all cities where a store is present.

$\pi_{\text{city}}(\text{store})$

**(b)** Find all stores in Bonn and Berlin. If not otherwise stated, the result should have all attributes of the specific relation, i.e. both `id` and `city` in this case.

$\sigma_{\text{city='Bonn'} \vee \text{city='Berlin'}}(\text{store})$

**(c)** Find all names of products sold in stores in Bochum.

$$\pi_{\text{name}}(\rho_{\text{product\_id}\leftarrow\text{id}}(\text{product}) \bowtie \text{sold\_in} \bowtie \rho_{\text{store\_id}\leftarrow\text{id}}(\pi_{\text{id}}(\sigma_{\text{city='Bochum'}}(\text{store}))))$$

**(d)** Which customers never ordered pizza?

IDs of customers who ordered pizza:

$$\rho_{\text{id}\leftarrow\text{customer\_id}}(\pi_{\text{customer\_id}}(\text{order} \bowtie \rho_{\text{product\_id}\leftarrow\text{id}}(\sigma_{\text{name='pizza'}}(\text{product}))))$$

Subtract from all customer ids and join with customers:

$$\text{customer} \bowtie (\pi_{\text{id}}(\text{customer}) - \rho_{\text{id}\leftarrow\text{customer\_id}}(\pi_{\text{customer\_id}}(\text{order} \bowtie \rho_{\text{product\_id}\leftarrow\text{id}}(\sigma_{\text{name='pizza'}}(\text{product})))))$$

**(e)** Which customers bought all the products?

$$\text{customer} \bowtie \rho_{\text{id}\leftarrow\text{customer\_id}}(\pi_{\text{customer\_id, product\_id}}(\text{order}) \div \pi_{\text{product\_id}}(\rho_{\text{product\_id}\leftarrow\text{id}}(\text{product})))$$

## Common Mistakes

- Lets consider the relations $A(U, V)$ and $B(X, Y)$ and the join $A \bowtie B$. Joining these two relations without common attributes will result in a cross product, which is probably not what you want. If you want to join over two differently named attributes, rename one of them to the name of the other and then use the natural join operator. For example, to join $A$ and $B$ where $U = X$, rename $X$ to $U$ and then join:

  $$A \bowtie (\rho_{\text{U}\leftarrow\text{X}}(B)) = C. \tag{1}$$

  The resulting relation $C(U, V, Y)$ will not have the attribute $X$ anymore because it has been renamed.

- Other definitions of relational algebra might define the join operator with a subscript. We have not defined such an operator, so if you want to use it, you have to provide your own definition. You also have to be very careful with the names of the attributes. For example, consider the relation $R(A, B)$ and the join.

  $$R \bowtie_{A=B} R. \tag{2}$$

  It is not clear whether the attributes $A$ and $B$ come from the left or right relation. In this case, the result is the same either way, but an example with different relations on both sides of the join operator is conceivable.

  Another issue is that it is not clear what the names of the attibutes of the result should be.

  To avoid issues with attribute name collisions, we can use the rename operator $\rho_{\text{B}\leftarrow\text{A}}(r)$ to rename the attribute $A$ of a relation $r$ to $B$.

  It is also possible to use "qualified" attribute names, where the attribute is prefixed with the name of the relation, but this is not always possible, as could be seen in the previous example, and often leads to issues later on where the qualified attribute names are forgotten.

- String values must be quoted since it would not be possible to differentiate them from attributes otherwise. The following query does not work because there is no attribute with the name "Banana".

$$\sigma_{\text{name}=\text{Banana}}(\text{product}). \tag{3}$$

To find all products with the name “Banana”, the value must be quoted:

$$\sigma_{\text{name}=\text{'Banana'}}(\text{product}). \tag{4}$$

- The division operator will consider **all** attributes of the relation on the left-hand side, even if you forgot they are there.