Databases

Relational algebra:

Operational – step-by-step computation of the result. Very useful for representing query execution plans.

Schema & syntax:

Example

Schema: **Sailors**(*sid*:`integer`, *name*: `string`, *age*: `real`)

Relation instance S of **Sailors**:

$S =$

sid	*name*	*age*
22	*Dustin*	35
31	*Lubber*	55.5
44	*Guppy*	35

Selection $\sigma_c(R)$:

Returns tuples (rows) of relation instance that satisfy the selection condition, i.e., c(a1, ..., an).

Example

$R =$

sid	*name*	*age*
22	*Dustin*	35
31	*Lubber*	55.5
44	*Guppy*	35

, $\sigma_{age<50}(R) =$

sid	*name*	*age*
22	*Dustin*	35
44	*Guppy*	35

, and

$\sigma_{age<50 \vee name=\text{'Lubber'}}(R) = R$, Here $\wedge$ =: 'and', $\vee$ =: 'or'.

Projection $\pi_{a_1,\ldots,a_n}(R)$:

Deletes fields (columns) that are not in projection list and then removes duplicate tuples.

Example

$R =$

sid	*name*	*age*
22	*Dustin*	35
31	*Lubber*	55.5
44	*Guppy*	35

,

$\pi_{age}(R) =$

age
35
55.5

and $\pi_{name,age}(R) =$

name	*age*
Dustin	35
Lubber	55.5
Guppy	35

Cross-product $R \times S$:

Allows us to combine two relations, we cross every row R with every row of S. We get $R_{\text{rows}} \cdot S_{\text{rows}}$

Example

$S =$

sid	name	age
22	Dustin	35
31	Lubber	55.5
44	Guppy	35

$, R =$

sid	bid	day
22	12	12/4/2013
31	4	8/4/2013

$S \times R =$

(sid)	name	age	(sid)	bid	day
22	Dustin	35	22	12	12/4/2013
31	Lubber	55.5	22	12	12/4/2013
44	Guppy	35	22	12	12/4/2013
22	Dustin	35	31	4	8/4/2013
31	Lubber	55.5	31	4	8/4/2013
44	Guppy	35	31	4	8/4/2013

Renaming $\rho_{i_1 \to a'_{i_1}, \ldots, i_k \to a'_{i_k}}$:

Renames fields using positional notation

Example

$$\rho_{1 \to sid1, 4 \to sid2}(S \times R) =$$

sid1	name	age	sid2	bid	day
22	Dustin	35	22	12	12/4/2013
31	Lubber	55.5	22	12	12/4/2013
44	Guppy	35	22	12	12/4/2013
22	Dustin	35	31	4	8/4/2013
31	Lubber	55.5	31	4	8/4/2013
44	Guppy	35	31	4	8/4/2013

Difference $R_1 \setminus R_2$:

The difference between R_1 and R_2.

Example

$S_1 =$

sid	name	age
22	Dustin	35
31	Lubber	55.5
44	Guppy	35

$, S_2 =$

sid	name	age
22	Dustin	35
31	Lubber	55.5

$S_1 \setminus S_2 =$

sid	name	age
44	Guppy	35

Union $R_1 \cup R_2$:

Returns a combination of R_1 and R_2

Intersection $R_1 \cap R_2$:

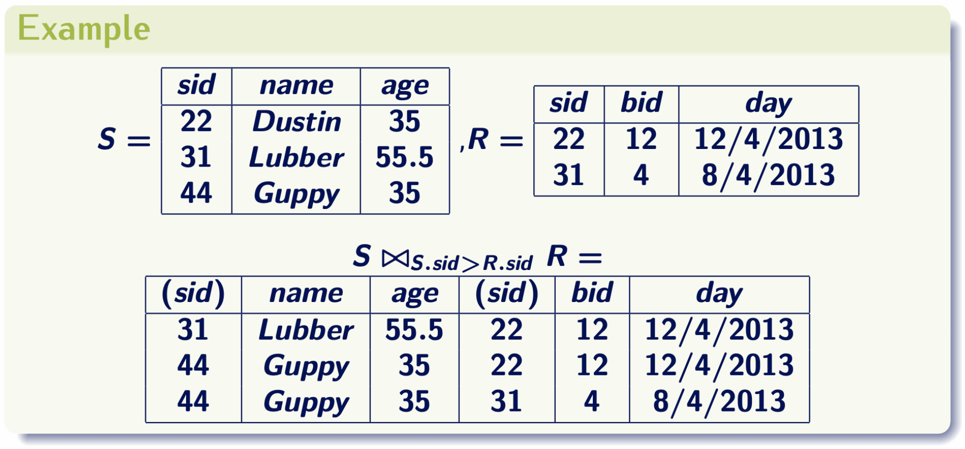
Intersection between 2 relations, meaning the rows both in R_1 and R_2

$$= R_1 \setminus (R_1 \setminus R_2)$$

Theta-Join $R \bowtie_{c(a_1,\ldots,a_n)} S$

Computes all combinations of tuples from R and S that satisfy condition $c(a_1, \ldots, a_n)$. In other words:

$$R \bowtie_{c(a_1,\ldots,a_n)} S = \sigma_{c(a_1,\ldots,a_n)}(R \times S)$$



Example

$S =$

sid	name	age
22	Dustin	35
31	Lubber	55.5
44	Guppy	35

$, R =$

sid	bid	day
22	12	12/4/2013
31	4	8/4/2013

$S \bowtie_{S.sid > R.sid} R =$

(sid)	name	age	(sid)	bid	day
31	Lubber	55.5	22	12	12/4/2013
44	Guppy	35	22	12	12/4/2013
44	Guppy	35	31	4	8/4/2013

Equi-Join $R \bowtie_{a_1,\ldots,a_k} S$:

Computes all combinations of tuples from R and S that satisfy condition $R.a_1 = S.a_1, \ldots, R.a_k = S.a_k$ and deletes duplicate columns (the first occurrence is kept):

$$R \bowtie_{a_1,\ldots,a_k} S = \sigma_{R.a_1=S.a_1,\ldots,R.a_k=S.a_k}(R \times S)$$

Natural Join $R \bowtie S$:

Computes equi-join on all common attributes of R and S. Columns with same name of associate relations will appear once only once (first occurrence in the positional notation).

Example

$S =$

sid	name	age
22	Dustin	35
31	Lubber	55.5
44	Guppy	35

$, R =$

sid	bid	day
22	12	12/4/2013
31	4	8/4/2013

$S \bowtie_{sid} R =$

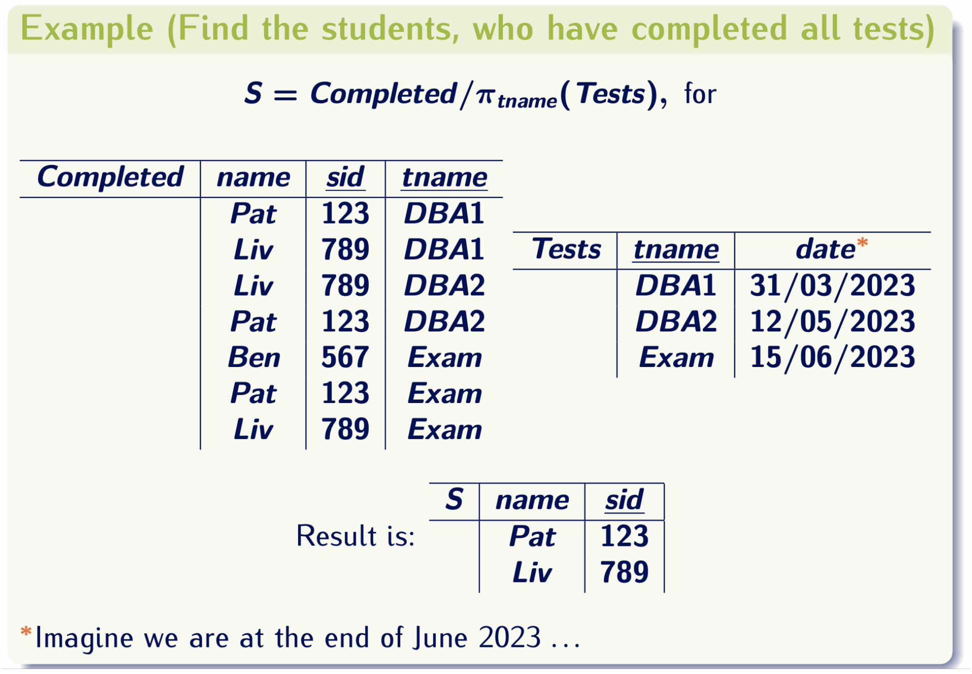
sid	name	age	bid	day
22	Dustin	35	12	12/4/2013
31	Lubber	55.5	4	8/4/2013

$S \bowtie R =$

sid	name	age	bid	day
22	Dustin	35	12	12/4/2013
31	Lubber	55.5	4	8/4/2013

Division R / S:

Find tuples in X that are related to all tuples in Y.
For all relations S and R it holds $(S \times R)$ / $R = S$.



Example (Find the students, who have completed all tests)

$$S = Completed / \pi_{tname}(Tests), \text{ for}$$

Completed	name	sid	tname
	Pat	123	DBA1
	Liv	789	DBA1
	Liv	789	DBA2
	Pat	123	DBA2
	Ben	567	Exam
	Pat	123	Exam
	Liv	789	Exam

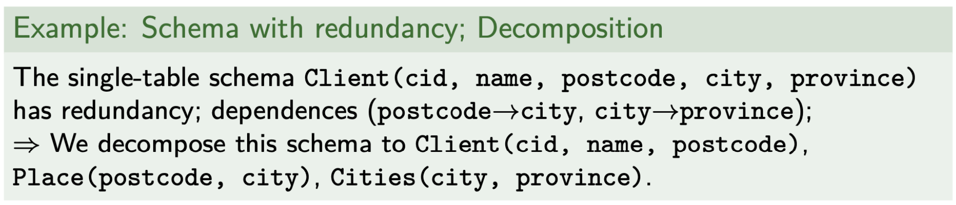
Tests	tname	date*
	DBA1	31/03/2023
	DBA2	12/05/2023
	Exam	15/06/2023

Result is:

S	name	sid
	Pat	123
	Liv	789

*Imagine we are at the end of June 2023 ...

Schema Refinement and Normal Forms



Example: Schema with redundancy; Decomposition

The single-table schema `Client(cid, name, postcode, city, province)` has redundancy; dependences (`postcode`→`city`, `city`→`province`);
⇒ We decompose this schema to `Client(cid, name, postcode)`, `Place(postcode, city)`, `Cities(city, province)`.

Functional dependence's:

Given 2 tuples in r, if the value X agrees, then Y must also agree. For example the **rating** of an employee that **determines** the **hourly wage**. You write this as:

$$R \to W$$

Where R is the **rating** and W is the **wage**.
This dependency **brings some problems** with it:

- What if we want to update the wage for everyone
- What happens when we insert an employee and don't know the wages for his rating
- If we delete all employees with rating 5, we loose all information for rating 5

To solve this we can use decomposition. **We put the wages and ratings in another table**. You can often refine a table to remove redundancy. For example:
Employees work in an department. All employees have a attribute named parking. Parking is shared between departments. It is better to move parking attribute to the departments.

Armstrong’s Axioms:

Reflexivity (Axiom of reflexivity):
If you have a set of attributes A, then $A \to A$.

Example: Suppose we have a table called “**Employees**” with attributes (columns) such as **EmployeeID**, **Name**, and **Department**. If we know that **EmployeeID determines EmployeeID** (which is always true since it is the primary key), we can say that **EmployeeID → EmployeeID** is a functional dependency.

Augmentation (Axiom of augmentation):
If $A \to B$, then $AC \to BC$ for any attribute C.

Example: Continuing with the “Employees” table, if we know that **EmployeeID determines Name**, we can use the axiom of augmentation to infer that **EmployeeID also determines both Name and Department**. So, we can say that EmployeeID → Name and EmployeeID → Department are functional dependencies.

Transitivity (Axiom of transitivity):
If $A \to B$ and $B \to C$, then $A \to C$.

Example: Consider a table called **“Courses” with attributes CourseID, Instructor, and Department**. If we know that **CourseID determines Instructor** (each course is taught by a specific instructor), and **Instructor determines Department** (an instructor belongs to a specific department), we can apply the axiom of transitivity to conclude that **CourseID determines Department**. Therefore, we have CourseID → Department as a functional dependency.

Union:
If $X \to Y$ and $X \to Z$, then $X \to YZ$

Decomposition:
If $X \to YZ$, then $X \to Y$ and $X \to Z$

Key and superkey

A superkey is a **set of attributes that uniquely identifies a record** in a table, while a minimal key is a **superkey that contains the smallest possible number of attributes** needed to uniquely identify a record.

Right-hand side rule:
The following functional dependencies (FDs) exist:
- OrderID → CustomerID, ProductID, OrderDate
- CustomerID → OrderDate

The right-hand side rule states that if an attribute never appears on the right-hand side of any FD in a table, then that attribute must be present in every key of the table.

In other words, in order to uniquely identify a record in the “Orders” table, we must include the attribute “OrderID” in every key. For instance, a key could be composed of {OrderID, CustomerID} or {OrderID, ProductID}, among other possibilities.

Left-hand side rule:
Let's say we have the following functional dependencies:

- EmployeeID $\to$ Name
- EmployeeID $\to$ Department
- Department $\to$ Salary

In our example, the attribute "Salary" never appears on the left-hand side of any functional dependency but appears on the right-hand side of the dependency Department $\to$ Salary. According to the left-hand side rule, we can conclude that "Salary" does not belong to any key of the "Employees" table.

This means that "Salary" is not necessary to uniquely identify a record in the table. It implies that two employees with the same EmployeeID, Name, and Department can have different salaries. Therefore, "Salary" is not a part of the key attributes for the "Employees" table.

Starting from small attribute sets:
Starting with small attribute sets (minimal keys) and excluding their proper supersets when searching for keys. This approach helps streamline the search process and ensures that we identify only the minimal keys that uniquely identify records in a table.

Normal Forms

Third normal form:
Every non-key attribute should dependent on the key, the whole key, and nothing but the key.

Decomposition:

Suppose we have a relation R with some attributes. A decomposition of R consists of two or more relations such that:

- Each new relation scheme contains a subset of the attributes of R
- Together the relations include all attributes in R

Example:
$\{AB, BCD, CD\}$ is a decomposition of $ABCD$, but $\{AB, BC\}$ is not because it does not include D.

Handy shit:

Check if R is in 3NF:

$R = \{A, B, C, D, E\}$

$A \rightarrow B$

$BC \rightarrow E$

$ED \rightarrow A$

Check if the letters in the right side are in the left side. Here, B, E and A are all in the left side so 3NF

Check if R is in BCNF:

$R = \{A, B, C, D, E\}$

$A \rightarrow B$

$BC \rightarrow E$

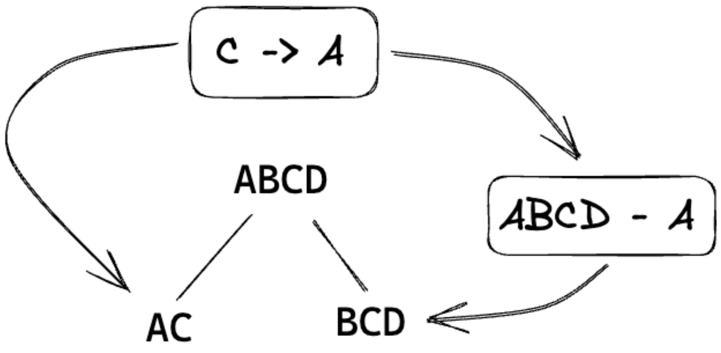
$ED \rightarrow A$

Check if the left side is a **Superkey**, meaning left should reach everything that is in R. For example EDC is a superkey, because $ED \rightarrow A$, and $A \rightarrow B$ meaning you can reach everything from EDC.

Decompose into BCNF:

First check if R is in BCNF. If it is not follow these steps:

1. Take all the dependence's that are not **Superkeys** of R and choose one.
2. Split the tree into the dependence and $R -$ the dependency. See the tree structure below:



3. Check if the branches are in BCNF, if not, repeat. Also check if dependencies are lost. If so, add these as new relations. For example:

$$A \rightarrow BC \text{ becomes the relation } ABC$$

If these are not in BCNF you are pretty much fucked and BCNF is not possible.