

IN3020&4020 – Database Systems (2022)

Part: Intro & Recap

Lecture 2: Relational Model

20 January

Egor V. Kostylev

IFI, University of Oslo

egork@ifi.uio.no



Lecture rules (reminder)

Digital (in *Zoom*) for some a couple of weeks:

- challenging for both **you** and **me**

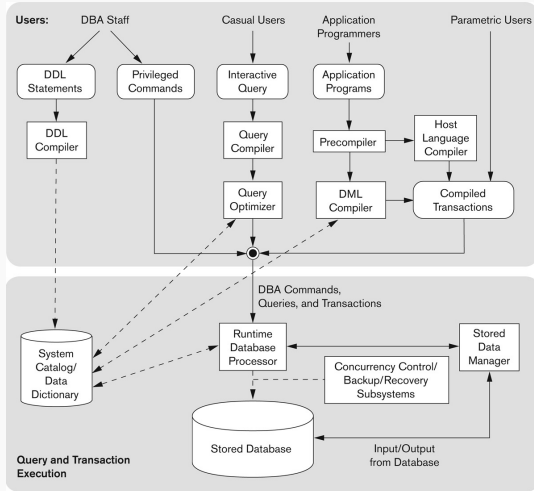
I need your **cooperation**:

- please, turn **video** on if possible
- ask **questions** (rise 'yellow' hand)
- I will also ask questions time to time

Lectures will be **recorded**:

- videos will be **published** (very) soon after the lecture
- do not forget about **mandatories** (see below)
- will **stop** recording when you speak (regulations)

Example DBMS architecture (Lecture 1 remainder)



Materials to read

1. *Fundamentals of Database Systems* by R. Elmasri and S.B. Navathe (7th edition): Chapter 5, Sections 5.1&5.2
2. *Foundations of Databases* by S. Abiteboul, R. Hull, and V. Vianu, Chapter 3
<http://webdam.inria.fr/Alice/>
3. *IN2090 – Databaser og datamodellering* (Leif Harald Karlsen, Dumitru Roman), Lecture 3
<https://www.uio.no/studier/emner/matnat/ifi/IN2090/h21/timeplan/>
4. *Wikipedia*, etc.

There are slight *variations* (we will discuss)

Lecture plan

1. History of data models
2. Classic relational model (a version of)
3. Integrity constraints (keys and foreign keys)
4. Difference with SQL data model (if have time)

1. History of data models

Database: collection of related **data**—that is, known facts with implicit meaning—organised in some way

Data model: the conceptual representation of data

- usually, **abstract** representation—that is, **hides** the storage and implementation details that are not important to usual database users

Brief history (look them up)

Early days (1960's) of digital data management:

- **Mainframe period**, 1 'big' machine for (that-time) 1M\$
- 512 Kbyte RAM, 50 Mbyte disk
- Other I/O-equipment magnetic & paper tape, punched-card reader and teletype (thus, unix has a funny abbreviation like `tty` for the «terminal»)
- **hierarchical** and **network** data models
- **ad-hoc** own query languages tailored for specific applications, largely **procedural** (i.e., not **declarative**)

Hierarchical database (actually, data) model

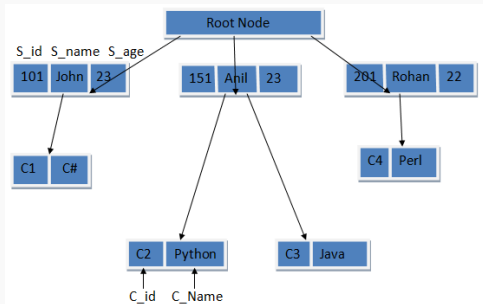
Underlies IBM's **Information Management System (IMS)** DBMS

Used in **bill of materials** for **Apollo** program

Data is represented in a **tree-like** structure

Edges are **parent-child** relationships

Own **query language** (called **DL/I**, tightly integrated into **COBOL**)



Network database model

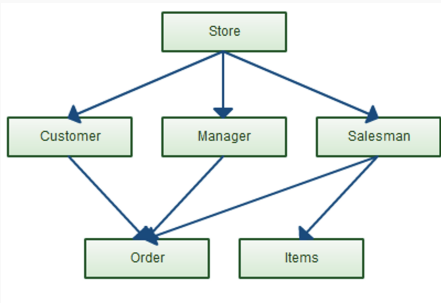
Underlies General Electric's **Integrated Data Store (IDS)** DBMS

Used by BT **Customer Service System** (and other projects)

Generalised hierarchical model: several parents of a child are allowed

Performance-oriented with low level of abstraction

Own (difficult to learn) **query language**



Codd's Relational Model

E. F. Codd, 1970: *A relational model of data for large shared data banks* (Commun. ACM 13, 6):

“Future users of large data banks must be protected from having to know how the data is organized in the machine (the internal representation). A prompting service which supplies such information is not a satisfactory solution. Activities of users at terminals and most application programs should remain unaffected when the internal representation of data is changed and even when some aspects of the external representation are changed. Changes in data representation will often be needed as a result of changes in query, update, and report traffic and natural growth in the types of stored information.”

What did Codd mean?

User **should not** see the internal representation (files, pointers):

- truly **abstract** data model

Relations as an abstract data structure:

- simple **universal** data model

Has a **declarative** query language **Relational Algebra** expressions:

- has **formal** theoretical foundations
- allows for **efficient** implementations: a lot of space for optimisation

Since then 'databases' usually mean 'relational databases':

- **SQL** is based on (variants of) relational model and algebra

Before we proceed to the details...

Since 2000 back to roots:

- XML is essentially a **hierarchical model**
- Knowledge Graphs (e.g., RDF) are essentially a **network model**

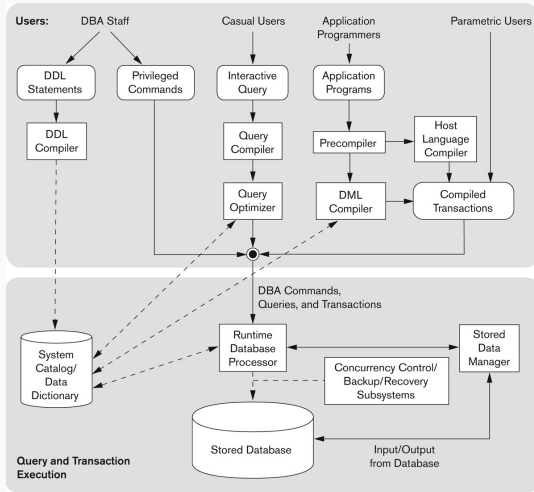
Question: Why they lost in the 70s, but regain popularity now?

Answer: Just *fancy*.

Real answer: we are on a **different level**:

- we **understand much better** what different models (and query languages) are good for
- these models have **universal** and **declarative** query languages (XQuery, SPARQL, Cypher, etc.)
- the languages are much more **efficient** in all components

The place of the relation model in DBMS architecture



Abstract model: this is what all the **users interact** with, but *not* how data is **stored in reality**

2. Classical relational model

Relations: intuition and terminology

A relation can be seen as a **table of values**. Something like this:

Tutorials:

<i>ID</i>	<i>site</i>	<i>tutorial</i>	<i>topic</i>
1	w3schools	SQL_2003STD	Databases
2	w3schools	HTML_5	WebDev
3	w3schools	CSS_3	WebDev
4	w3resource	SQL_2003STD	Databases
5	w3resource	MySQL	Databases

- **Tutorials:** **relation name**
- ID, site, tutorial, topic: **attribute names** (or just **attributes**), collectively: **signature**
- Relation name + signature: **relation schema** (or just **relation**)
- Rows: **tuples**, **records**, or **instances**, collectively: **relation state**
- Elements of tuples: **attribute values** (or just **attributes** as well)

Relations: properties

Relation

Tutorials:

<i>ID</i>	<i>site</i>	<i>tutorial</i>	<i>topic</i>
1	w3schools	SQL_2003STD	Databases
2	w3schools	HTML_5	WebDev
3	w3schools	CSS_3	WebDev
4	w3resource	SQL_2003STD	Databases
5	w3resource	MySQL	Databases

- Each attribute has a **domain** (i.e., a set of potential values): integers, strings, etc.
- Different attributes may have the same domain
- Attribute values **are from** the associated domain
- Tuples are **unordered** and **unique** (i.e., not identical)

Relations: properties

Same relation

Tutorials:

<i>ID</i>	<i>site</i>	<i>tutorial</i>	<i>topic</i>
2	w3schools	HTML_5	WebDev
3	w3schools	CSS_3	WebDev
1	w3schools	SQL_2003STD	Databases
4	w3resource	SQL_2003STD	Databases
5	w3resource	MySQL	Databases

- Tuples are **unordered** and **unique** (i.e., not identical)

Not a relation(!)

Tutorials:

<i>ID</i>	<i>site</i>	<i>tutorial</i>	<i>topic</i>
1	w3schools	SQL_2003STD	Databases
2	w3schools	HTML_5	WebDev
1	w3schools	SQL_2003STD	Databases
3	w3schools	CSS_3	WebDev
4	w3resource	SQL_2003STD	Databases

- Tuples are **unordered** and **unique** (i.e., not identical)

Relations: properties

Also same relation

Tutorials:

<i>site</i>	<i>ID</i>	<i>tutorial</i>	<i>topic</i>
w3schools	1	SQL_2003STD	Databases
w3schools	2	HTML_5	WebDev
w3schools	3	CSS_3	WebDev
w3resource	4	SQL_2003STD	Databases
w3resource	5	MySQL	Databases

- Tuples are **unordered** and **unique** (i.e., not identical)
- Attributes are also **unordered** and **unique**
(not so essential, but convenient for us;
may be different in different formalisations)

Formally: relation schema

Domain: a set of values

- may be **finite** (e.g., countries {*Norway*, *UK*, *Russia*, ...}) or **infinite** (e.g., integers)
- **atomic**—that is, values have no essential internal structure (e.g., not sets or tuples themselves)
- usually, has a corresponding **datatype** in the query language

Attribute (i.e., ‘column name’): a name with an associated domain

- domain of attribute A is written $dom(A)$
- for example, $dom(ID)$ is positive integers

Relation schema: a relation name R (just string) and several attributes $A_1, A_2, \dots, A_n$

- written as $R[A_1, A_2, \dots, A_n]$ (or $R[A_2, A_1, \dots, A_n]$, order is irrelevant)
- n is called the **arity** of R (**question:** can the arity be 0?)
- for example $Tutorials[ID, site, tutorial, topic]$

Formally: tuples

Tuple (or **record**, **instance**) over relation schema $R[A_1, \dots, A_n]$:
assignment of attribute values $v_1 \in \text{dom}(A_1), \dots, v_n \in \text{dom}(A_n)$ to
attributes $A_1, \dots, A_n$:

- formally, can be written $A_1 \rightarrow v_1, \dots, A_n \rightarrow v_n$
- we usually write $R\langle v_1, \dots, v_n \rangle$ or even $\langle v_1, \dots, v_n \rangle$
- example tuples over $\text{Tutorials}[ID, site, tutorial, topic]$:

$\langle 1, w3schools, SQL_2003STD, Databases \rangle$

$\langle 2, w3schools, HTML_5, WebDev \rangle$

Convenient notation (for later): if t is a tuple over $R[A_1, \dots, A_n]$
and $\{A'_1, \dots, A'_m\} \subseteq \{A_1, \dots, A_n\}$ then $t[A'_1, \dots, A'_m]$ is the
projection of t to $\{A'_1, \dots, A'_m\}$:

$\langle 2, w3schools, HTML_5, WebDev \rangle[ID, topic] = \langle 2, WebDev \rangle$

When $m = 1$, we can write $t.A$ instead of $t[A]$

Formally: relations and databases

Relation state over relation schema $R[A_1, \dots, A_n]$: a finite **set** I of tuples over $R[A_1, \dots, A_n]$

- ‘set’ means no order of elements, no repetitions
- may be written $I \subseteq \text{dom}(A_1) \times \dots \times \text{dom}(A_n)$
(but formally not correct for us due to the order mismatch)

Relation: a relation schema + relation state

Database: finite set of relations (i.e., essentially, tables) with different names

Variants of the relational model

Many technical **non-essential variations** can be found:

- unordered vs. **ordered** signature (i.e., attributes, columns)
- named perspective (for attributes) vs. **unnamed**:
 - if the signature is ordered, names of attributes are **not important** and may be ignored (not the case for tables!)
 - moreover, attribute domains may be **abstracted** away (i.e., only one big domain is considered)
 - relation schema is just **name and arity** (written as R^2 or $R(\cdot, \cdot)$)
 - inspired by **formal logic** (which is foundations for databases):
tuples are logical **facts** $R(a, b)$

Other details are (very) **essential**:

- no order, no repetitions of tuples in a relation, etc. (see below)

Semantics (the meaning) of databases

Intuitive: **tuples** represent **certain facts** that correspond to a real-world relationships

Tutorials(1, *w3schools*, *SQL_2003STD*, *Databases*)

means that there is a **SQL_2003STD** tutorial on **Databases** topic on **w3schools** site with ID **1** (of course, the fact may not hold, but the database creators **claim** it)

Closed world assumption is assumed in the relational databases: all facts **not mentioned** in the database, are claimed **not to hold**.

No tuple *Tutorials*(1, *w3schools*, *Mars*, *Astronomy*)—no tutorial.

Why to bother? Allows to intuitively answer many queries (e.g., queries with **NOT EXISTS**)

Different in other database models: **RDF** knowledge graphs, etc.

3. Integrity constraints

Key integrity constraints in a relation: intuition

Tutorials:

<i>ID</i>	<i>site</i>	<i>tutorial</i>	<i>topic</i>
1	w3schools	SQL_2003STD	Databases
2	w3schools	HTML_5	WebDev
3	w3schools	CSS_3	WebDev
4	w3resource	SQL_2003STD	Databases
5	w3resource	MySQL	Databases

Attribute *ID* is **special** in this relation: has **no repeated** values

- called a **candidate** key (of the relation)

Question: any other candidate keys in this relation?

Not a **coincidence**: we *want* it to be a real ID

- can declare *ID* to be **primary** key of the schema

Tutorials[*ID*, *site*, *tutorial*, *topic*] so that every relation over this schema has *ID* as a candidate key

Candidate and super keys formally

Super key of a relation (schema + state): a subset X of the attributes of the relation such that if t_1 and t_2 are two different tuples then $t_1[X] \neq t_2[X]$

- **observation**: the relation signature is **always** a super key

Candidate key of a relation: a **minimal** super key

- that is, removing any attribute causes the remaining attributes to no longer be a super key
- sometimes, candidate keys are **underlined**:

Tutorials:

<u>ID</u>	site	tutorial	topic
1	w3schools	SQL_2003STD	Databases
2	w3schools	HTML_5	WebDev
...			

Candidate and super key examples

Tutorials:

<i>ID</i>	<i>site</i>	<i>tutorial</i>	<i>topic</i>
1	w3schools	SQL_2003STD	Databases
2	w3schools	HTML_5	WebDev
3	w3schools	CSS_3	WebDev
4	w3resource	SQL_2003STD	Databases
5	w3resource	MySQL	Databases

Super keys: $\{ID\}$, $\{ID, site\}$, $\{site, tutorial\}$,
 $\{ID, site, tutorial, topic\}$, etc.

Not super keys: $\{site\}$, $\{tutorial, topic\}$, etc.

Candidate keys: $\{ID\}$, $\{site, tutorial\}$ **Not unique!**

Not candidate keys: $\{ID, site\}$, $\{tutorial, topic\}$, etc.

Primary keys formally

Extend our relational model notions:

Relation schema with a primary key: a relation name R , several attributes $A_1, A_2, \dots, A_n$, and a subset of $A_1, A_2, \dots, A_n$ called **primary key**

- written as $R[\underline{\underline{A_1}}, A_2, \dots, \underline{\underline{A_n}}]$, where the double underlined attributes are the primary key (single underline is also used)
- only **one** primary key per relation schema is allowed
- for example $Tutorials[\underline{\underline{ID}}, site, tutorial, topic]$

Relation state over $R[\underline{\underline{A_1}}, A_2, \dots, \underline{\underline{A_n}}]$ is a relation state over $R[A_1, A_2, \dots, A_n]$ with the primary key of $R[\underline{\underline{A_1}}, A_2, \dots, \underline{\underline{A_n}}]$ being a super key

Question: Do you remember how to create a primary key in SQL?

Primary key examples

Tutorials:

<i>ID</i>	<i>site</i>	<i>tutorial</i>	<i>topic</i>
1	w3schools	SQL_2003STD	Databases
2	w3schools	HTML_5	WebDev
3	w3schools	CSS_3	WebDev
4	w3resource	SQL_2003STD	Databases
5	w3resource	MySQL	Databases

Relation over *Tutorials*[ID, *site*, *tutorial*, *topic*] ($\{ID\}$ is a super key)

Not relation over *Tutorials*[*ID*, *site*, *tutorial*, *topic*] ($\{site\}$ is not a super key)

Relation over *Tutorials*[*ID*, *site*, *tutorial*, *topic*]

Observation: since all attributes are always a super key, we can assume that if a primary key for a schema is not given, then it is silently all the attributes

Foreign key (referential) integrity constraints intuition

Tutorials:

<i>ID</i>	<i>site</i>	<i>tutorial</i>	<i>topic</i>
1	w3schools	SQL_2003STD	Databases
2	w3schools	HTML_5	WebDev
3	w3schools	CSS_3	WebDev
4	w3resource	SQL_2003STD	Databases
5	w3resource	MySQL	Databases

Students:

<i>name</i>	<i>tutorialID</i>
Alice	1
Alice	2
Bob	2

Values of attribute *tutorialID* are **not arbitrary** in relation *Students*:
all values are assumed to be mentioned in *ID* of *Tutorials*

- can declare *tutorialID* to be a **foreign key** in *Students*[*name*, *tutorialID*] **referring** to the primary key *ID* of *Tutorials*[*ID*, *site*, *tutorial*, *topic*] so that the two relations in the database are required to be consistent

This notion generalises to **attribute sets**

Foreign key (referential) constraints formally

Sorry, a bit **technical**; but the idea is simple

Foreign key attributes $B'_1, \dots, B'_k$ of a relation schema

$S[B_1, \dots, B_m]$ referring to attributes $A'_1, \dots, A'_k$ of relation

$R[A_1, \dots, A_n]$: attributes such that

$$\text{dom}(B'_1) = \text{dom}(A'_1), \dots, \text{dom}(B'_k) = \text{dom}(A'_k)$$

Relations over $S[B_1, \dots, B_m]$ and $R[A_1, \dots, A_n]$ **satisfy** this foreign

key if every tuple t_1 over $S[B_1, \dots, B_m]$ **refers** to a tuple t_2 over

$R[A_1, \dots, A_n]$ —that is, $t_1[B'_1, \dots, B'_k] = t_2[A'_1, \dots, A'_k]$

A database (set of relations) can come with **foreign keys**, which should be **satisfied**

Foreign key (referential) constraints examples

Tutorials:

<i>ID</i>	<i>site</i>	<i>tutorial</i>	<i>topic</i>
1	w3schools	SQL_2003STD	Databases
2	w3schools	HTML_5	WebDev
3	w3schools	CSS_3	WebDev
4	w3resource	SQL_2003STD	Databases
5	w3resource	MySQL	Databases

Students:

<i>name</i>	<i>tutorialID</i>
Alice	1
Alice	2
Bob	2

Satisfy the foreign key *tutorialID* of *Students* referring to *ID* of *Tutorials*

Foreign key (referential) constraints examples

Tutorials:

<i>ID</i>	<i>site</i>	<i>tutorial</i>	<i>topic</i>
1	w3schools	SQL_2003STD	Databases
2	w3schools	HTML_5	WebDev
3	w3schools	CSS_3	WebDev
4	w3resource	SQL_2003STD	Databases
5	w3resource	MySQL	Databases

Students:

<i>name</i>	<i>tutorialID</i>
Alice	1
Alice	2
Bob	2
Bob	25

Do not satisfy the foreign key *tutorialID* of *Students* referring to *ID* of *Tutorials*

Foreign keys observations

Sometimes, foreign keys are allowed to refer **only to primary keys**

- see our example

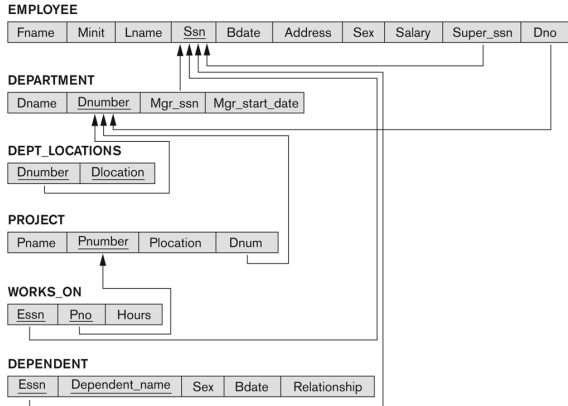
Corresponding attributes **do not need to** have the same name

- see our example

It is allowed to refer to the same table

- imagine a table of employees with a *managerID* attribute (the big boss has to manage himself/herself)

Complex schema example



Copyright © 2016 Ramez Elmasri and Shamkant B. Navathe

Foreign keys are convenient to draw with **arrows**

In this figure, all foreign keys consist of **one attribute** (simple case)

Who needs to care about keys?

Question: Does a casual database user who writes a query over a database needs to know about keys?

Answer: Not really

Question: Does a database administrator who inputs data into a database needs to know about keys?

Answer: Yes, of course

Question: Does a DBMS programmer who writes a (`SELECT`) query engine needs to bother about keys?

Answer: Yes, because keys are the knowledge that may help a lot to evaluate queries efficiently, and we will see how

Difference with SQL data model

SQL is based on the **relational model**, but there are some **differences** between its model and the relational model

Question: What are the main differences?

Answer: two main differences

1. **NULLs**
2. repeated and ordered tuples in **output** relations

SQL NULLs

In SQL, there is a special value `NULL`, that can appear in tuples as usual domain values

The meaning of `NULL` is manifold:

- unknown, not applicable, not exists, etc.

The effect on the data model is minor:

- just assume that `NULL` belongs to all domains of attributes
- special constraint on attributes can be used to forbid `NULL` values
- usually, `NULL` is forbidden for primary keys
- foreign keys have a special treatment of `NULL`

SQL NULLs

In **SQL**, there is a special value **NULL**, that can appear in tuples as usual domain values

The meaning of **NULL** is manifold:

- **unknown**, **not applicable**, **not exists**, etc.

The effect on the query language is **dramatic**:

- very **non-intuitive** and causes a lot of **confusion** for all types of users (e.g., are two **NULLs** equal or not?)
- requires a lot of special non-trivial treatment in **implementation** and often slows down **performance**
- we **will not** pay much attention to **NULL** since there is **nothing** conceptually interesting for our studies of the **main principles**

In the passing on NULLs

In 2009, Sir C. A. R. (or Tony) Hoare said:

"I call it my billion-dollar mistake. It was the invention of the null reference in 1965. At that time, I was designing the first comprehensive type system for references in an object oriented language (ALGOL W). My goal was to ensure that all use of references should be absolutely safe, with checking performed automatically by the compiler. But I couldn't resist the temptation to put in a null reference, simply because it was so easy to implement. This has led to innumerable errors, vulnerabilities, and system crashes, which have probably caused a billion dollars of pain and damage in the last forty years."

Bag (multiset) relations example

Tutorials:

<i>ID</i>	<i>site</i>	<i>tutorial</i>	<i>topic</i>
1	w3schools	SQL_2003STD	Databases
2	w3schools	HTML_5	WebDev
3	w3schools	CSS_3	WebDev
4	w3resource	SQL_2003STD	Databases
5	w3resource	MySQL	Databases

SQL query `SELECT Tutorials.site FROM Tutorials` gives

π_{site} **Tutorials:**

<i>site</i>
w3schools
w3schools
w3schools
w3resource
w3resource

Input: relation. Output: **not relation.**

Question: Why?

Bag (multiset) relations motivation 1

SQL input: relations. SQL output: **not necessarily relation**.

Question: Why?

Answer 1: duplicate elimination is computationally expensive

- if a user does not care, better to keep the duplicates
- if they care, they can use DISTINCT

`SELECT DISTINCT Tutorials.site FROM Tutorials` gives

$\delta(\pi_{site} \mathbf{Tutorials})$:
<u>site</u>
w3schools
w3resource

Bag (multiset) relations motivation 2

SQL input: relations. SQL output: **not necessarily relation**.

Question: Why?

Answer 2: duplicates are essential for aggregate queries

```
SELECT COUNT(*) AS cnt FROM  
    SELECT Tutorials.site FROM Tutorials
```

$$\frac{\frac{\gamma_{COUNT(*) \rightarrow cnt}(\pi_{site} \mathbf{Tutorials})}{cnt}}{5}$$

Input & output: **relations**. Intermediate: **not relation**

Essential to define (compositional) semantics

- in this particular query, we can rewrite the an equivalent query that manipulates only relations
- this is not always the case **Homework:** Can you come up to an example?

Bag (multiset) relational model formally

What we need to change in our formalisation? Surprisingly little:

Bag relation state over relation schema $R[A_1, \dots, A_n]$: a finite **bag** I of tuples over $R[A_1, \dots, A_n]$

- ‘bag’ (a.k.a. **multiset**) means no order of elements, but repetitions are allowed (**observation**: every set is a bag)
- $\{a, b, c\}$ is a set, $\{\{a, a, b, c\}\} = \{\{a, b, a, c\}\}$ is a bag

Bag relation: a relation schema + bag relation state

Bag database: finite set of bag relations with different names

- usual relations are sometimes called **set** relations to contrast

Observation: super keys (including candidate keys) are possible only for set relations

- cannot silently assume that every relation schema has a primary key (for sets, it is all attributes)

Bag relations: examples

Relation (and bag relation)

Tutorials:

<i>ID</i>	<i>site</i>	<i>tutorial</i>	<i>topic</i>
1	w3schools	SQL_2003STD	Databases
2	w3schools	HTML_5	WebDev
3	w3schools	CSS_3	WebDev
4	w3resource	SQL_2003STD	Databases
5	w3resource	MySQL	Databases

Bag relations: examples

Not a relation, but bag relation

Tutorials:

<i>ID</i>	<i>site</i>	<i>tutorial</i>	<i>topic</i>
1	w3schools	SQL_2003STD	Databases
2	w3schools	HTML_5	WebDev
1	w3schools	SQL_2003STD	Databases
3	w3schools	CSS_3	WebDev
4	w3resource	SQL_2003STD	Databases

Bag relations: examples

Same bag relation

Tutorials:

<i>ID</i>	<i>site</i>	<i>tutorial</i>	<i>topic</i>
1	w3schools	SQL_2003STD	Databases
1	w3schools	SQL_2003STD	Databases
2	w3schools	HTML_5	WebDev
4	w3resource	SQL_2003STD	Databases
3	w3schools	CSS_3	WebDev

Ordered tables in SQL

Tutorials:

<i>ID</i>	<i>site</i>	<i>tutorial</i>	<i>topic</i>
1	w3schools	SQL_2003STD	Databases
2	w3schools	HTML_5	WebDev
3	w3schools	CSS_3	WebDev
4	w3resource	SQL_2003STD	Databases
5	w3resource	MySQL	Databases

SELECT * FROM Tutorials ORDERED BY topic gives

OrderedTutorials:

<i>ID</i>	<i>site</i>	<i>tutorial</i>	<i>topic</i>
1	w3schools	SQL_2003STD	Databases
4	w3resource	SQL_2003STD	Databases
5	w3resource	MySQL	Databases
2	w3schools	HTML_5	WebDev
3	w3schools	CSS_3	WebDev

Output is the **same** relation as input,
but it seems that we are **missing** something...

Ordered relations (and ordered bag relations) in SQL

One may argue that `ORDER BY` only influences the **output**, and can be **ignored** on the intermediate levels of query evaluation

If so, we could essentially abstract away this '**presentational**' issue

Unfortunately, this is **not true**. Query

```
SELECT DISTINCT Tutorials.topic FROM
```

```
    SELECT * FROM Tutorials ORDERED BY topic LIMIT 3
```

gives only *Databases*, but

```
SELECT DISTINCT Tutorials.topic FROM
```

```
    SELECT * FROM Tutorials ORDERED BY topic DESC LIMIT 3
```

gives both *Databases*, *WebDev*.

Do we need to reconsider our data model again? **Yes**(

Ordered relations (and ordered bag relations) in SQL

Do we need to reconsider our data model again? Yes(

But we **will not** in this course:

- such queries are **rare** and **not very meaningful**
- ordered relational model is **technical**, but **straightforward**
- this model is **not essential** for the rest of this course
- we mostly concentrate on **set relational model** as the most **fundamental** for databases, mentioning **bags** when necessary

What have we learned?

The foundations for any study of relational databases:

Relational model