Set Operations (III)

- Query 9: Find the identifiers of professors who teach “ethics” *but not* “maieutics”.

```
(select p.pers-id from professors p, lectures l
 where p.pers-id = l.held_by and l.title = ‘ethics’)
except
(select p.pers-id from professors p, lectures l
 where p.pers-id = l.held_by and l.title = ‘maieutics’);
```

- Schema compliant table schemas required as operands

- Operations **union**, **except**, and **intersect** produce tables as sets of tuples, elimination of duplicates

- Operations **union all**, **except all**, and **intersect all** maintain duplicates in result table

Set Operations (IV)

- Semantics of the set operations extended by the clause **all**: Let $F(R, t)$ describe the frequency of tuple t in table R. Then the number of duplicates with respect to the tables R and S is defined as:
  - $\forall\, t : F(R \textbf{ union all } S, t) = F(R, t) + F(S, t)$
  - $\forall\, t : F(R \textbf{ except all } S, t) = \textbf{if } F(R, t) \geq F(S, t) \textbf{ then } F(R, t) - F(S, t) \textbf{ else } 0$
  - $\forall\, t : F(R \textbf{ intersect all } S, t) = \min(F(R, t), F(S, t))$

Null Values (I)

- Each SQL data types contains a special value **null**
- The special value **null** for an attribute value in a tuple indicates the value is *unknown* or *unclear* (e.g., the age of a person)
- Null values present special problems in relational operations including arithmetic operations, comparison operations, and set operations
- The result of an arithmetic expression (involving, e.g., +, –, *, /) is null if any of the input values is null
  - Example 1: Expression r.A + 7: If r.A is null for a particular tuple, the result of the expression is null
  - Example 2: Expression r.A * s.B: If any of the two arguments, or both arguments have the value null for particular tuples *r* and *s*, the result is null

Null Values (II)

- Comparisons involving null values are a bigger problem
- Does the comparison "5 < **null**" yield *true* or *false*?
  - To say this is true is wrong since we do not know what the null value represents
  - To say this is false is also wrong since then the expression "**not** (5 < **null**)" would evaluate to true, which does not make sense
  - Therefore, SQL adds a third truth value **unknown** that is the result of any comparison involving a *null* value
- SQL uses a *three-valued logic* with the values **true**, **false**, and **unknown**
- Boolean operations **and**, **or**, and **not** are extended:

not	
true	false
unknown	unknown
false	true

and	true	unknown	false
true	true	unknown	false
unknown	unknown	unknown	false
false	false	false	false

Null Values (III)

or	true	unknown	false
true	true	true	true
unknown	true	unknown	unknown
false	true	unknown	false

- In the **where** clause only those tuples are added to the result where the filter condition yields *true*
- The condition "**where** A **is null**" allows to select all tuples with a *null* value in attribute A
- The condition "**where** A **is not null**" allows to select all tuples that do not have a *null* value in attribute A
- Two copies of a tuple, such as t_1 = ('A', null) and t_2 = ('A', null), are considered to be equal, even if some attributes have a *null* value, although a comparison "null = null" would usually return *unknown*, rather than *true*
- This also holds for the set operations **union**, **intersect**, and **except**

String Operations (I)

- In SQL strings are enclosed in *single* quotes, for example, ‘database’
- Equality operation on strings is case sensitive, for example, the expression ‘database’ = ‘DATABASE’ yields false
- Examples of string operations
  - String concatenation performed by “||”, e.g., the expression ‘database ’ || ‘system’ yields ‘database system’
  - Extracting substrings with the method *substring*, e.g., the expression **substring**(‘database’, 3, 4) yields the string ‘taba’
  - Finding the length of strings with the method *length*, e.g., the expression **length**(‘database’) yields the number 8
  - Converting strings to uppercase (lowercase) with the method *upper* (*lower*), e.g., the expression **upper**(‘database’) yields the string ‘DATABASE’
- Unfortunately, different DBS offer different sets of string operations; further, the same string operations can have a slightly different syntax

String Operations (II)

- Pattern matching
  - Percent (%) symbol: It matches any substring
  - Underscore (_) symbol: It matches any character
- Examples
  - 'Intro%' matches any string beginning with "Intro"
  - '%Comp%' matches any string containing "Comp" as a substring
  - '___' matches any string of exactly three characters
  - '___%' matches any string of at least three characters
- Patterns are expressed in an SQL query by the **like** comparison operator
- Example: Find all students with names Meyer, Meier, Maier, Mayer, etc.

```
select *
from students
where name like 'M__er';
```

Ordering the Display of Tuples (I)

- Sorted output of query results is frequently required
- Examples
  - Entries in a telephone book
  - List of students in a course, sorted by last name (higher priority) and first name (lower priority)
  - Faculty of a university, ordered by college name, department name, last name, and first name
- Sorting large volumes of data is expensive since it requires an external sorting operator (e.g., external merge sort)
- Sorting is the last step in an SQL query and is performed by the **order by** clause with respect to one or more attributes
  - Syntax: **order by** [**asc** | **desc**] A_1, ..., [**asc** | **desc**] A_n A_i attribute
  - Sorting order: **asc** = ascending, **desc** = descending
  - Attributes have a different sorting priority; it decreases from the attribute A_1 with the highest priority to A_n with the lowest priority

Ordering the Display of Tuples (II)

- Query example: Determine the personnel identifier, name, and rank of all professors. (1) Sort the result tuples in descending order by rank and in ascending order by name. (2) Sort the result tuples in ascending order by name and in descending order by rank.

  (1) **select** pers-id, name, rank **from** professors **order by** rank **desc**, name **asc**;

  (2) **select** pers-id, name, rank **from** professors **order by** name **asc**, rank **desc**;

pers-id	name	rank
2136	Curie	C4
2137	Kant	C4
2126	Russel	C4
2125	Sokrates	C4
2134	Augustinus	C3
2127	Kopernikus	C3
2133	Popper	C3

(1)

pers-id	name	rank
2134	Augustinus	C3
2136	Curie	C4
2137	Kant	C4
2127	Kopernikus	C3
2133	Popper	C3
2126	Russel	C4
2125	Sokrates	C4

(2)

Modification of the Database

- So far: Attention to the extraction (search) of information from the database
- Now: Manipulation of the database
- Three SQL statements available
  - **insert** – adds new rows of data to a table
  - **update** – modifies existing data in a table
  - **delete** – removes rows of data from a table

Insertion of Tuples into a Table (I)

- Insertion of a tuple with constant attribute values
- Syntax: **insert into** <relation name>[(<attribute name> [, <attribute name>]*)]
  values (<constant> [, <constant>]*);
- Examples
  - **insert into** professors **values** (2136, ‘Curie’, ‘C4’, 536);
    Input of attribute values according to the order in the schema definition
  - **insert into** professors (pers-id, name, rank, room)
    values (2136, ‘Curie’, ‘C4’, 536);
    Same as before but with the corresponding list of attributes
  - **insert into** professors (rank, pers-id, room, name)
    values (‘C4’, 2136, 536, ‘Curie’);
    Different order of attributes and attribute values

Insertion of Tuples into a Table (II)

- Examples (*continued*)
  - **insert into** professors **values** (2136, 'Curie' , **null**, 536);

    Rank is unknown, attribute *rank* should and does *not* have the constraint "**not null**"

  - **insert into** professors (pers-id, name, rank, room)

    values (2136, 'Curie', **null**, 536);

    Same as before but with a list of attributes

  - **insert into** professors (room, pers-id, name)

    values (536, 2136, 'Curie');

    It is possible to insert only a part of the attribute values of a tuple, if, e.g., some values are unknown. The undefined fields are automatically filled by the system with *null* values if they are allowed to contain them.

Insertion of Tuples into a Table (III)

- Generation of tuples by means of a query
- Syntax: **insert into** <relation name>[(<attribute name> [, <attribute name>]*)]
  select … **from** … **where** …;
- Examples
  - **insert into** attends **select** reg-id, id
    from students, lectures
    where title = ‘logic’;
  - **insert into** students
    select * **from** students;
    - Before inserting any tuple, the SQL query has to be fully evaluated
    - Statement would duplicate every tuple in the *students* relation if the relation did not have a primary key constraint

Update of Tuples in a Table (I)

- Changing values in a tuple without changing *all* values in a tuple
- Syntax: **update** <relation name>

  set <attribute name> = <expression>

  [, <attribute name> = <expression>]*

  [**where** <condition>];
- The **where** clause of the **update** statement may contain any construct legal in the **where** clause of the **select** statement
- Examples
  - Increase the semester number of each student by 1

    update students **set** sem = sem + 1;
  - Change room number (currently 232) of professor Russel to 115

    update professors **set** room = 115 **where** name = 'Russel';

    Assumption: We assume that only one professor Russel exists

Update of Tuples in a Table (II)

- The construct **case** is used to perform several updates in a single **update** statement

  Syntax: **case**
  when <predicate$_1$> **then** <result$_1$>
  …
  when <predicate$_n$> **then** <result$_n$>
  else result$_0$
  end;

- Example: Reorganization of the individual assistants' offices into open-plan offices

```
update assistants
set room = case
             when room >= 100 and room < 120 then 417
             when room >= 120 and room < 140 then 438
             else 455
           end;
```

Deletion of Tuples in a Table

- Only whole tuples of a single table can be deleted with a single command
- Syntax: **delete from** <relation name> [**where** <condition>];
- The **where** clause of the **delete** statement may contain any construct legal in the **where** clause of the **select** statement
- Examples
  - Delete students who study longer than 8 semesters

    delete from students **where** sem > 8;
  - Delete all *tests* tuples

    delete from tests;
  - Lecture “foundations” with the identifier 5001 was suddenly canceled; delete all registrations

    delete from attends **where** id = 5001;

Aggregate Functions (I)

- So far: Attention to the
  - extraction (search) of information from a database (determines a subset of the stored data in the database, based on a filter condition)
  - manipulation (insertion, update, deletion) of a database

- Now:
  - Calculation of new results from the data stored in the database
  - The new results have not been explicitly stored in the database before

- Example
  - We store each student's semester number in the database
  - But we do not keep the average semester calculated over all students in the database in order to see whether the student population is young, middle-aged, or old

Aggregate Functions (II)

- Aggregate functions are functions that take a collection (list) of values as input and return a *single* value as output
- The collection of values consists of the values either of a whole column or of a part of a column
- SQL offers 5 built-in aggregate functions
  - **count** returns the *number* of values in a specified column
  - **avg** returns the *average* of the values in a specified column
  - **sum** returns the *sum* of the values in a specified column
  - **min** returns the *minimum* of the values in a specified column
  - **max** returns the *maximum* of the values in a specified column
- Input to **count**: collection of values of any SQL data type
- Input to **avg** and **sum**: collection of numbers only
- Input to **min** and **max**: collections of values of any SQL data type with a defined order relation '<'

Aggregate Functions (III)

- Each aggregate function (except **count**) eliminates null values first from the collection and operates only on the remaining non-null values

- The term **count**(*) indicates a special use of **count**; it counts all the rows of a table, regardless of whether null values or duplicate tuples occur

- The keyword **distinct**
  - placed in front of the column name in the aggregate function is used to eliminate duplicates before the aggregate is computed
  - has only an effect on the result of the functions **count**, **avg**, and **sum**

- Aggregate functions can *only* be used in the **select** clause and the **having** clause (see the later discussion of grouping)

Aggregate Functions (IV)

- Query 1: Calculate the number of students.

  select count(*) **from** students;

- Query 2: Calculate the number of students, and store the result in a new attribute named “total”.

  select count(*) **as** total **from** students;

- Query 3: Calculate the number of students with different names.

  select count(**distinct** name) **from** students;

- Query 4: Calculate the number of different semesters the students are in, and store the result in a new attribute named “diff_sem”.

  select count(**distinct** sem) **as** diff_sem **from** students;

8

Query 1

total
8

Query 2

8

Query 3

diff_sem
7

Query 4

Aggregate Functions (V)

- Query 5: Calculate the number of students in the second or eight semester.

  select count(*) **from** students **where** sem = 2 **or** sem = 8;

- Query 6: Calculate the average semester of all students, and store the result in a new attribute named “avg_sem”.

  select avg(sem) **as** avg_sem **from** students;

- Query 7: Compute the average grade for all performed tests so far.

  select avg(grade) **from** tests;

- Query 8: Calculate the total of credits for all lectures, and store the result in a new attribute named “total_cred”.

  select sum(credits) **as** total_cred **from** lectures;

3

Query 5

avg_sem
7.625

Query 6

1.66

Query 7

total_cred
30

Query 8

Aggregate Functions (VI)

- Query 9: Determine the lowest room number of assistants.

  select min(room) **from** assistants;

- Query 10: Determine the highest semester number of a student, and store the result in a new attribute named “max_sem”.

  select max(sem) **as** max_sem **from** students;

- Query 11: Compute the largest difference of student semester numbers.

  select max(sem) – **min**(sem) **from** students;

- Query 12: Determine the lowest grade (new attribute “low_grade”) and the highest grade (new attribute “high_grade”) of tests.

  select max(grade) as low_grade, **min**(grade) as high_grade **from** tests;

101

Query 9

max_sem
18

Query 10

16

Query 11

low_grade	high_grade
2	1

Query 12