Information Systems

Chapter 3:

SQL

Rita Schindler | TU Ilmenau, Germany
www.tu-ilmenau.de/dbis

Outline

- Basic Queries in SQL
- Nested Queries
- Aggregation & Grouping
- Recursive Queries
- Data Definition

Example Database

CUSTOMERS (FName, LName, CAddress, Account)

PRODUCTS (Prodname, Category)

SUPPLIERS (SName, SAddress, Chain)

orders ((FName, LName) → CUSTOMERS, SName → SUPPLIERS, Prodname → PRODUCTS, Quantity)

offers (SName → SUPPLIERS, Prodname → PRODUCTS, Price)

Basic Structure

- SQL is based on set and relational operations with certain modifications and enhancements.
- A typical SQL query has the form

select $A_1, A_2, \dots, A_n$
from $r_1, r_2, \dots, r_k$
where P

  - A_i represent attributes
  - r_i represent relations
  - P is a predicate
- this query is equivalent to the relational algebra expression

$$\Pi_{A1,\,A2,\dots,\,An}(\sigma_P(r_1 \times r_2 \times \dots \times r_k))$$

Basic Structure (2)

- The result of an SQL query is a relation (set of tuples) with a schema defined through the attributes A_is.
- The **select** clause corresponds to the projection operation of the relational algebra; it is used to list the attributes to be output in a query result.
- Example: Find the name of all suppliers.

```
select SName
from SUPPLIERS ;
```


Basic Structure (3)

- An asterisk "*" in the **select** clause denotes all attributes

```
select * from SUPPLIERS;
```

- SQL allows duplicate tuples in a relation as well as in query results. Duplicates can be removed from query result using keyword **distinct**

```
select distinct Account from CUSTOMERS;
```

- **select** clause can contain arithmetic expressions as well as functions on attributes including attributes and constants.

```
select substr(SName,1,10) as "Name",
       Prodname, Price * 100
from offers;
```


Basic Structure (4)

- The **where** clause corresponds to the selection operation of the relational algebra. It consists of a predicate involving attributes of the relations that appear in the **from** clause.
- Example: List the first and last name of customers having a negative account.

```
select FName, LName
from CUSTOMERS
where Account < 0 ;
```

- Logical connectives **and**, **or**, and **not** can be used to formulate complex condition in **where** clause.

Basic Structure (5)

- In SQL you always have to specify the join condition explicitly (or use explicit **natural join** operators)!!!
- Example: List the name and address of suppliers that offer products. Remove duplicates from the result and list the result ordered by the supplier's address.

```
select distinct SUPPLIERS.SName, SAddress
from SUPPLIERS, offers
where SUPPLIERS.SName = offers.SName
order by SAddress;
```

Set Operations

- The SQL set operations `union`, `except`, and `intersect` correspond to the relational algebra operations ∪, − , ∩.
- Each of the above operations automatically eliminates duplicates. To retain duplicates for the `union` operator, one has to use the corresponding multiset version `union all`.
- Example: Find all suppliers that offer a SkyPhone or SuperPhone.

```
(select SName from offers
 where Prodname = 'SkyPhone')
union
(select SName from offers
 where Prodname = 'SuperPhone') ;
```


Nested Subqueries (1)

- So far, `where` clauses in examples only consist of simple attribute and/or constant comparisons.
- SQL provides language constructs for the nesting of queries using subqueries.
- A subquery is a `select-from-where` expression that is nested within another query.

Nested Subqueries (2)

- Most common use of subqueries is to perform tests for set membership, set comparisons, and set cardinality.
- Set valued subqueries in a `where` condition:
  - *expression* `[not]in` *(subquery)*
  - *expression comparison-operator* `any` *(subquery)*
  - *expression comparison-operator* `all` *(subquery)*
- Set cardinality or test for (non-)existence:
  - `[not]exists` *(subquery)*
- Subqueries in a `where` clause can be combined arbitrarily using logical connectives.

Examples of Set Valued Subqueries

- Give the name and chain of all suppliers located in Kazan that offer a SkyPhone for less than $500.

```
select SName, Chain
from SUPPLIERS
where SName in(select SName
               from offers
               where Prodname = 'SkyPhone'
               and Price < 500 )
and SAddress like '%Kazan%';
```

- This query can also be formulated using a join!

Examples of Set Valued Subqueries (2)

- Find the name and address of customers who have ordered a product from Macrosoft.

```
select *
from CUSTOMERS
where(FName,LName) in (select FName, LName
                       from orders
                  where Sname ='Macrosoft');
```


Examples of Set Valued Subqueries (3)

- Find all customers from Moscow who have an account greater than any (some) customer in Kazan.

```
select *
from CUSTOMERS
where Account > any (select Account
                     from CUSTOMERS
                     where CAddress like '%Kazan%')
and CAddress like '%Moscow%';
```

- Note that **= any** is equivalent to **in**.

Examples of Set Valued Subqueries (4)

- Example: List all customers who have an account greater than all customers from Kazan.

```
select *
from CUSTOMERS
where Account > all (select Account
                     from CUSTOMERS
                     where CAddress like'%Kazan%');
```

- Note that **<> all** is equivalent to **not in**.

Examples of Set Valued Subqueries (5)

- Give all suppliers (SName) who offer at least one product cheaper than all other suppliers.

```
select SName
from offers O1
where Price <= all (select Price
                    from offers O2
                    where O1.Prodname = O2.Prodname
                    and O1.SName <> O2.SName ) ;
```

- If a subquery refers to attributes of an outer query, the subquery is called a correlated subquery.
- References to outer relations and attributes typically occur through using aliases.

Test for (non-)existence

- List all customers who have ordered a product from a supplier in Kazan.

```
select *
from CUSTOMERS C
where exists(select *
             from orders O, SUPPLIERS S
             where O.SName = S.SName
             and O.FName = C.FName
             and O.LName = C.LName
             and SAddress like '%Kazan%');
```


Test for (non-)existence (2)

- Give all products (Prodname, Category) for which no offer exists.

```
select *
from PRODUCTS P
where not exists ( select *
                   from offers
                   where P.Prodname = Prodname);
```

- Alternatively:

```
select *
from PRODUCTS P
where Prodname not in ( select Prodname
                        from offers);
```


NULL Values

- If permitted by the schema definition for a table (i.e., no not null constraints), attributes can have null values.
- null ≙ unknown, non-existent, or non-applicable value
- Result of any arithmetic expression involving null is null. Result of **where** clause condition is false if it evaluates to null.

and	true	false	null
true	true	false	null
null	null	false	null
false	false	false	false

or	true	false	null
true	true	true	true
null	true	null	null
false	true	false	null

not	
true	false
null	null
false	true

Null Values (2)

- Examples: Give all suppliers that are not associated with a chain.

```
select *
from SUPPLIERS
where Chain is null;
```

- List all customers who have a known account.

```
select *
from CUSTOMERS
where Account is not null;
```

- All aggregate functions except count(*) ignore tuples with null values on the aggregate attribute(s).

Aggregate Functions

- Aggregate functions operate on a multiset of values and return a single value. Typical aggregate functions are `min`, `max`, `sum`, `count`, and `avg`.
- For aggregate functions (and the following grouping), an extension of relational algebra exists.
- Examples: What is the total number of suppliers?

```
select count(SName)
from SUPPLIERS;
```

- How many different products are offered?

```
select count (distinct Prodname)
from offers;
```


Grouping

- Idea: Group tuples that have the same properties into groups, and apply aggregate function to each group.
- Optionally, consider only groups for the query result that satisfy a certain group condition.
- Syntax in SQL:

select *attribute(s) [with aggregate function]*
from $R_1, R_2, \ldots, R_m$
[**where** *P*]
group by *grouping attribute(s)*
[**having** *condition on group*];

Grouping: Examples

- Examples: For each supplier, list the name of the supplier and the total number of products the supplier offers.

```
select SName, count(Prodname)
from offers
group by SName;
```

- For each customer, list the total quantity of orders.

```
select FName, LName, sum(Quantity)
from orders
group by FName, LName;
```

- Note: attributes that appear in the select clause outside of an aggregate function must appear in the **`group by`** clause !

Grouping (cont.)

- A query containing a **`group by`** clause is processed in the following way:
  1. Select all rows that satisfy the condition specified in the **`where`** clause.
  2. From these rows form groups according to the **`group by`** clause.
  3. Discard all groups that do not satisfy the condition in the **`having`** clause.
  4. Apply aggregate function(s) to each group.
  5. Retrieve values for the columns and aggregations listed in the **`select`** clause.

Example: Lakes and Countries

Lake	Depth	Country
Bodensee	252	Germany
Tschadsee	7	Tschad
Bodensee	252	Switzerland
Goldsee	1435	Eldorado
Gardasee	346	Italy
Tschadsee	7	Niger
Vaenernsee	100	Sweden
Titicacasee	272	Peru
Tanganjikasee	1435	Zaire
Tschadsee	7	Nigeria
Tanganjikasee	1435	Tansania
Silbersee	272	Peru
Tanganjikasee	1435	Burundi
Eduardsee	117	Zaire
Tanganjikasee	1435	Sambia
Titicacasee	272	Bolivia
Victoriasee	85	Uganda
Eduardsee	117	Uganda
Victoriasee	85	Kenia
Genfer See	310	Switzerland
Victoriasee	85	Tansania
Ontariosee	236	USA
Baikalsee	1620	Russia
Schatzsee	272	Phantasia
Tanasee	72	Ethiopia
Ontariosee	236	Canada

Country	Continent
Sweden	Europe
Kenia	Africa
USA	America
Ethiopia	Africa
China	Asia
Tschad	Africa
Switzerland	Europe
Peru	America
Italy	Europe
Niger	Africa
Germany	Europe
Phantasia	Antarctica
Russia	Europe
Nigeria	Africa
Zaire	Africa
Bolivia	America
Tansania	Africa
Mexico	America
Burundi	Africa
Australia	Australia
Sambia	Africa
Canada	America
Uganda	Africa
Eldorado	America

Example: Lakes and Countries - Queries

Solve the following queries in SQL

- Which lakes are located in Europe? Give lakes and their depths.
- List all continents with their inherited lakes.
- Count the number of lakes per continent.
- In which continent reside at least five lakes?
- In how many countries is each lake located?
- Which is the greatest depth of all lakes?
- Which is the deepest lake of all?
- Which lakes (including depth) are located in Africa?
- Which is the deepest lake in Africa?
- Give for every country in Africa its deepest lake (if there is one) and order by depth!
- Give the average depth of all lakes!

Common Table Expressions

- Query expression, which is referenced in the query multiple times
- Syntax

with *query-name* [(*list-of-columns*)] **as** (*query expression*)

- Query without **with**

```
select *
from offers
where Price * 1.1 <= ( select avg(Price)
                       from offers )
and Price * 0.9 >= ( select avg(Price)
                     from offers )
```


Common Table Expressions (cont.)

- query with **with** clause

```
with AVERAGE(AvgPrice) as
  ( select avg(Price)
    from offers )
select *
from offers, AVERAGE
where Price * 1.1 <= AvgPrice
and Price * 0.9 >= AvgPrice
```

Recursive Queries

- Usage: Bill of Material queries, calculating transitive closure (flight connection etc.)
- Example:

Departure	Arrival	FlightTime
FRA	LHR	2
FRA	JFK	8
LHR	SFO	10
SFO	HNL	4
FRA	AMS	2

Recursive Queries (cont.)

- Flight connections with at most two changes

```
select Departure, Arrival
from FLIGHT_CONNECTION
where Departure = 'FRA'
   union
select F1.Departure, F2.Arrival
from FLIGHT_CONNECTION F1, FLIGHT_CONNECTION F2
where F1.Departure = 'FRA'
and F1.Arrival = F2.Departure
   union
select F1.Departure, F3.Arrival
from FLIGHT_CONNECTION F1, FLIGHT_CONNECTION F2,
     FLIGHT_CONNECTION F3
where F1.Departure = 'FRA'
and F1.Arrival = F2.Departure
and F2.Arrival = F3.Departure
```


Recursion in SQL:2003

- Query formulation using an extended **with recursive** query
- Syntax

```
with recursive recursion-table as (
   query-expression -- recursive part
)
[ traversal-clause ] [ cycle-clause ]
query-expression -- non-recursive part
```

- non-recursive part: query on recursion table

Recursion in SQL:2003 (cont.)

- recursive part

```
-- initialization
select ...
from table where ...
-- recursion step
union all
select ...
from table, recursion-table
where recursion-condition
```

Recursion in SQL:2003: Example

```
with recursive TRIP(Departure, Arrival) as
 ( select Departure, Arrival
   from FLIGHT_CONNECTION
   where Departure = 'FRA'
      union all
   select T.Departure, F.Arrival
   from TRIP T, FLIGHT_CONNECTION F
   where T.Arrival = F.Departure )
select distinct *
from TRIP
```


Recursion: Examples (cont.)

- arithmetic operation during recursion steps

```
with recursive TRIP(Departure, Arrival, TotalTime) as
( select Departure, Arrival, FlightTime as TotalTime
  from FLIGHT_CONNECTION
  where Departure = 'FRA'
      union all
  select T.Departure, F.Arrival, TotalTime+FlightTime
  from TRIP T, FLIGHT_CONNECTION F
  where T.Arrival = F.Departure  )
select distinct *
from TRIP
```


Safety of Recursive Queries

- Safety (= finiteness of computation) is an important requirement to query languages
- Problem:
  - Cycles in recursion, e.g. insert a new flight connection from Honolulu(HNL) to London Heathrow(LHR)
- Addressed in SQL by:
  - restricting depth of recursion
  - cycle detection

Safety of Recursive Queries (cont.)

- Restricting depth of recursion

```
with recursive TRIP(Departure, Arrival, Changes) as
 ( select Departure, Arrival, 0
   from FLIGHT_CONNECTION
   where Departure = 'FRA'
      union all
   select T.Departure, F.Arrival, Changes + 1
   from TRIP T, FLIGHT_CONNECTION F
   where T.Arrival = F.Departure
   and Changes < 2  )
select distinct *
from TRIP
```

Safety by Detecting Cycles

- Cycle clause
  - when duplicates are detected in path of computation of *attrib*: CycleColumn = '*' (Pseudo column of type **char**(1))
  - ensures finiteness of result "by hand"

```
cycle attrib set mark to '*' default '-'
```


Safety by Detecting Cycles (cont.)

```
with recursive TRIP(Departure, Arrival, Path) as
 ( select Departure, Arrival,
          Departure || '-' || Arrival as Path
   from FLIGHT_CONNECTION
   where Departure = 'FRA'
      union all
   select T.Departure, F.Arrival,
          T.Path || '-' || F.Arrival as Path
   from TRIP T, FLIGHT_CONNECTION F
   where T.Arrival = F.Departure  )
   cycle Arrival set FoundCycle to '*' default '-'
select Path, FoundCycle
from TRIP
```


Safety by Detecting Cycles (cont.)

Path	FoundCycle
FRA-LHR	-
FRA-JFK	-
FRA-AMS	-
FRA-LHR-SFO	-
FRA-LHR-SFO-HNL	-
FRA-LHR-SFO-HNL-LHR	*

Data Definition Language (DDL)

- Allows the specification of not only a set of relations but also information about each relation, including
  - the schema of a relation
  - the domain of attributes
  - integrity constraints
  - the set of indexes associated with a relation (later)
  - the physical storage structure of a relation (later)

Data Types in SQL

- `char(n)`, `varchar2(n)` (in SQL standard only `varchar(n)`)
- `number(m,n)`, `real`, `int`, `smallint`, ...
- `long`, `date`

Creating a Table

- Syntax:

```
create table name (
   attribute1 datatype [not null]  [unique]
        [attribute constraint] ,
   ...
   attribute2  datatype  [not null]  [unique]
        [attribute constraint] ,
    [table constraint(s)]
);
```


Integrity Constraints

- **not null** (do not allow null values)
- **primary key** *attribute* (as attribute constraint)
- **primary key** (*list of attributes*) (as table constraint)
- **unique** *attribute* (as attribute constraint)
- **unique** (*list of attributes*) (as table constraint)
- **check** *condition*
  - If *condition* only refers to one attribute → attribute constraint;
  - if *condition* includes more than one attribute of the relation → table constraint;
  - *condition* must be a simple condition that does not contain queries or references to other relations!

Integrity Constraints

Foreign key (or referential integrity) constraints:

- **references** *relation[.attribute]* → attribute constraint
- **foreign key** *attributes* **references** *relation[.attributes]* → table constraint

- In many DBMS (e.g. Oracle, PostgreSQL) each constraint can be named using **constraint** *name constraint-specification*

Example (PostgreSQL)

```
create table Students (
   StID numeric constraint Students_pk primary key,
   FName varchar(50) not null,
   LName varchar(50) not null,
   DOB date constraint dob_check
            check( DOB is not null
                   and to_char(DOB) > '01-JAN-01' ),
   Major char(5) constraint fk_majors references Majors,
   ZipCode integer constraint check_zip
                  check(ZipCode is not null
                        and ZipCode between 1 and 99999),
   City varchar(50),
   Street varchar(50),
   Started date not null,
   constraint dates_check check( DOB < Started ),
   constraint name_add unique( FName, LName, DOB) );
```


Modifications of the Database: Deletions

- Syntax:
  delete from *relation* [**where** *condition*];
- Example: Delete all suppliers that don't offer any product.

```
delete from SUPPLIERS
where SName not in (select SName
                    from offers);
```


Modifications of the Database: Deletions

- Examples (cont.): Delete all customers having an account less than the average account of all customers.

```
delete from CUSTOMERS
where Account < (select avg(Account)
                 from CUSTOMERS);
```

- Problem: Evaluating the condition after each deletion of a customer tuple leads to a change of the subquery result.
- In SQL: First compute **avg**(Account) and identify tuples from CUSTOMERS to delete; then delete those tuples without recomputing **avg**(Account).

Modifications of the Database: Insertions

- Add the customer Scott Tiger (who is living in Ilmenau).

```
insert into CUSTOMERS
values('Scott','Tiger','Ilmenau',null);
```

$\triangleq$

```
insert into
CUSTOMERS(FName, LName, CAddress, Account)
values('Scott','Tiger','Ilmenau',null);
```

or

```
insert into
CUSTOMERS(FName, LName, CAddress)
values('Scott','Tiger','Ilmenau');
```

Modifications of the Database: Updates

- Increase the account of the customer Scott Tiger by $5,000, and change his address to Erfurt.

```
update CUSTOMERS
set Account = Account+5000, CAddress = 'Erfurt'
where LName='Tiger' and FName='Scott';
```

- Set Clark Kent's account to the account of Scott Tiger.

```
update CUSTOMERS
set Account = ( select Account
                from CUSTOMERS
                where LName='Tiger'
                and FName='Scott')
where FName='Clark'
and LName='Kent';
```


Summary

- SQL – standard query language for relational database systems
- basic query structure = relational core of SQL
- extensions to relational algebra:
  - aggregation and grouping
  - common table expressions and recursive queries
- syntactic sugar, views, . . .