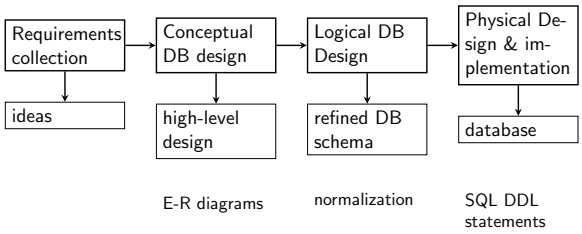
Databases (LIX022B05)
SQL basics

Instructor: Çağrı Çöltekin
c.coltekin@rug.nl
Information science/Informatiekunde

2012-09-24

Database Design Process



E-R diagrams | normalization | SQL DDL statements

Conceptual (E-R) design: things to remember

- Entity / entity set
- Relationship / relationship set
- Attribute
  - Simple
  - Composite
  - Multi-valued
- Weak entity
- one-to-one, one-to-many, many-to-one relationships
- total or partial participation
- binary or n-ary relationships
- Converting E-R diagrams to table schemas
- Primary keys, foreign keys
- SQL **create table** statement

DB schema refinement/normalization: things to remember

- Database anomalies
  - Insertion anomaly
  - Deletion anomaly
  - Update anomaly
- First normal form (1NF)
- Functional dependencies
- Third normal form (3NF)
- Boyce-Codd normal form (BCNF)
- Decomposition
  - Lossless-join
  - Dependency preserving
- Multi valued dependencies
- Fourth normal form (4NF)

Some guidelines on DB design

- Clear semantics: it should be easy to explain the meaning of your DB schema.
  - use understandable names
  - do not overload your tables
- Avoid redundancy and inconsistency: a good DB design should prevent all anomalies. Follow the most strict normal form you can.
- Joins should reference only keys.
- Avoid null values.

This week

This week we will go through a selective set of frequently used SQL commands.

- Data definition commands: **create table**, **alter table**.
- Data manipulation commands: **insert into**.
- Queries
  - a brief introduction to relational algebra
  - SQL **select** statement, with simple joins.

SQL: a few introductory remarks

- SQL is 'the' database query, definition and modification language in the DB industry.
- It has been around since 1970's (initially it was called SEQUEL).
- Even though it is commonly expanded to 'Structured Query Language', SQL is more than a query language. As well as query capabilities, SQL includes
  - commands to define a database (DDL: data definition language),
  - commands to manipulate data in the database (DML: data manipulation language)
- A cautionary note: even though SQL is an ISO standard,
  - most vendors only implement a subset of the ISO standard.
  - there are many non-standard extension implemented by various vendors.

SQL as data definition language

We have already experimented with,

- **create table**
- **alter table**
- **drop table**

SQL can also be used to create, drop or alter other objects, such as,

- databases,
- views,
- triggers,
- stored functions/procedures,
- indexes

SQL: create table

General form:

```
create table table_name (attr1  type1   [constraints],
                         ...    ...     [constraints],
                         attrN typeN    [constraints],
                         constraints, ...);
```

An example

```
create table order(ID int  not null  unique,
                   ISBN char(13) not null,
                   street  varchar(50),
                   postcode char(6),
                   city  varchar(50)
                         default 'Groningen',
                   cID int  check (cID > 0),
                   primary key(ID),
                   foreign key(ISBN) references  book(ISBN),
                   foreign key(cID) references   customer(ID));
```


SQL: data types

char(n)	a character string of n characters.
varchar(n)	a character string at most n characters.
int(n)	integer value, with an optional number of digits. Note that the number of digits only affects how the number is displayed.
numeric(n,m)	a fixed point (real) number.
float(n)	a floating point (real) number. (also real and double with machine-dependent precision).
time,date,datetime	as the names suggest, time, date or both together.
blob	a possibly big chunk of data with no identifiable structure.
text	a large piece of text.

SQL: constraints

Per-attribute constraints

not null	attribute is not allowed to take null values.
unique	attribute has to be unique.
default	attribute takes the default value if no value supplied.

Table-wide constraints

primary key	A chosen set of attributes that uniquely identify each row in the table.
foreign key	A set of attributes that are primary key of another table.
check	states allowed values for the attribute.

Support for constraints widely diverge from DBMS to DBMS.

SQL: foreign keys

Foreign keys are used to preserve the integrity of the data in the database. The full form of constraint is:

```
foreign key(attr)  references
    f_table_name(f_attr)
    on delete action
    on update action
```

Where action can be,

restrict	do not allow actions that violate referential integrity
no action	the same as restrict, but check is done during the execution of the action.
cascade	propagate the change to the referencing columns.
set null	set the referencing attributes to null.
set default	set the referencing attributes to their defaults.

Note: default action is DBMS specific

SQL: alter table

The **alter table** statement is the way to fix your mistakes during crate **table**.

Examples:

```
alter table customers rename to customer;
alter table customer add column points int default 0;
alter table customer drop column city;
alter table customer change column postcode pcode char(6);
alter table customer drop primary key;
alter table customer add primary key (ID);
```


SQL as data manipulation language

delete, **insert** and **update** statements change the data in the database.

Examples:

```
insert into customer
        values (2, 'some name', 'A–weg 30',
                '9718CW', 'Groningen', 5555, 0);
insert into customer (id,street)   values (2, 'A–weg 30');
delete from customer where id = 1;
delete from customer;
update customer set pcode='9719CW'where ID = 1;
update customer set pcode='9718CW';
```

Some other statements, for example **select** ... **into**, also modifies the data.

Relational Algebra

- The queries in SQL closely follows the formal query language relational algebra.
- The knowledge of relational algebra comes handy even for a practitioner (for example, to understand query optimization).
- Relational algebra defines a set of operations on relations (tables).
- Relational algebra operations take one or more relations, and return a relation.

Select operation (σ)

The relational algebra select operator σ_P selects a set of tuples (rows) from a relation where the condition 'P' is met.

book

bID	pages	year
1	130	1995
2	544	1995
3	213	2005
4	210	2012

Examples:

- $\sigma_{year=1995}$(book) :

bID	pages	year
1	130	1995
2	544	1995

- $\sigma_{pages>300}$(book) :

bID	pages	year
2	544	1995

The select operation corresponds to 'where' clause of the SQL queries, not the 'select' clause.

Project operation (π)

The relational algebra project operator π_{list} selects a set of attributes (columns) from a relation listed in *list*.

Example: $\pi_{bID,year}(\text{book})$:

book		
bID	**pages**	**year**
1	130	1995
2	544	1995
3	213	2005
4	210	2012

bID	**year**
1	1995
2	1995
3	2005
4	2012

Composing select and project

Examples:

book		
bID	**pages**	**year**
1	130	1995
2	544	1995
3	213	2005
4	210	2012

$\pi_{bID,year}(\sigma_{year=1995}(\text{book}))$:

bID	**year**
1	1995
2	1995

$\sigma_{pages>300}(\pi_{bID,pages}(\text{book}))$:

bID	**pages**
2	544

Note that the order of operations is sometimes significant.

Cartesian product ($\times$)

book		
bID	**pages**	**year**
1	130	1995
2	544	1995
3	213	2005
4	210	2012

orders		
cID	**bID**	**qty**
1	1	1
1	2	1
3	1	3
4	3	1

student × advisor					
bID	**pages**	**year**	**cID**	**bID**	**qty**
1	130	1995	1	1	1
1	130	1995	1	2	1
1	130	1995	3	1	3
1	130	1995	4	3	1
2	544	1995	1	1	1
2	544	1995	1	2	1
2	544	1995	3	1	3
2	544	1995	4	3	1
3	213	2005	1	1	1
3	213	2005	1	2	1
3	213	2005	3	1	3
3	213	2005	4	3	1
4	210	2012	1	1	1
4	210	2012	1	2	1
4	210	2012	3	1	3
4	210	2012	4	3	1

Natural join ($\bowtie$)

Natural join ($\bowtie$) operation combines rows from two tables where the common attributes match. It can be expressed as,

$$r \bowtie s = \sigma_{s.A_1=r.A_1 \wedge s.A_2=r.A_2 \cdots s.A_n=r.A_n}(r \times s)$$

where $A_1 \ldots A_n$ are the attributes common to both relations.

book		
bID	**pages**	**year**
1	130	1995
2	544	1995
3	213	2005
4	210	2012

orders		
cID	**bID**	**qty**
1	1	1
1	2	1
3	1	3
4	3	1

book ⋈ orders				
bID	**pages**	**year**	**oID**	**qty**
1	130	1995	1	1
1	130	1995	3	3
2	544	1995	1	1
3	213	2005	4	1

Other relational operations

- Set operations
  - Set union ($\cup$)
  - Set intersection ($\cap$)
  - Set difference ($-$)

  Set operations are typically applied to results of other operations (σ, π).
- Outer join
  - left outer join (⟕)
  - right outer join (⟖)
  - full outer join (⟗)

  Outer join operations are useful when there are **null** values in relations to be joined.

We will return to these operations.

SQL: queries

General form:

select attribute$_1$, $\cdots$, attribute$_N$
from table_name$_1$, $\cdots$, table_name$_M$
where condition;

- It is generally more intuitive to read and write SQL queries starting from **from**, then **where** and at last **select**.
- **from** lists the tables from which the columns/rows to be selected.
- **where** specifies which rows to be selected (relational algebra select operation (σ)).
- **select** lists the columns to be displayed (relational algebra project operation (π)).

Note that result of every SQL query is a single table (relation).

Simple query examples

book		
bID	**pages**	**year**
1	130	1995
2	544	1995
3	213	2005
4	210	2012

select $*$ **from** book;

bID	**pages**	**year**
1	130	1995
2	544	1995
3	213	2005
4	210	2012

select bID,**year** **from** book;

bID	**year**
1	130
2	544
3	213
4	210

select $*$ **from** book **where** bID $= 1$;

bID	**pages**	**year**
1	130	1995

select bID **from** book
where bID > 2 **and year** $= 1995$;

bID
2

select $*$ **from** book
where $10*$pages $>$ **year**;

bID	**pages**	**year**
2	544	1995
3	213	2005
4	210	2012

SQL and truth values

SQL **where** clause states a condition (predicate) that evaluates to either **true** or **false** (more on this when we discuss **null** values).

- Logical operations: **and**, **or** and **not**.
- Equality: $=$, inequality: $<>$, greater than: $>$, less than: $<$, greater than or equal to: $>=$, less than or equal to: $<=$.
- Arithmetic operations can be used if necessary. For example endt $-$ startt > 10, or hours / $40 > 1$
- The precedence of operators are generally what you expect: arithmetic first, comparison operators next, and logical connectives last. When in doubt: use parentheses ().

These are the basics, we will see more.

SQL string operations

- Beware of equality check involving **char** and **varchar**. Results may be unexpected due to padding.
- Order of character strings are based on lexicographic order ('a' < 'b').
- SQL **like** phrase allows comparison of parts of strings.
  - underscore '_' matches any character.
  - percent sign '%' matches any substring of size 0 or more.

  Examples:
  - **like** 'A%' matches any string that starts with an 'A' (including 'A').
  - **like** '__' matches any string with two characters.
  - **like** '__%' matches any string with two or more characters.
- **upper**() and **lower**() functions convert given string to upper or lowercase characters.
- multiple character strings can be concatenated using **concat**().

Renaming attributes

customer

cID	street_addr	pcode	city	points
1	A-weg 30	9718CW	Groningen	1
2	Broerstraat 5	9712CP	Groningen	1
3	OBS 34	9712GK	Groningen	1
4	Nijenborgh 9	9747AG	Groningen	3

```
select cID as 'ID',
       concat(street_addr, ' ', pcode, ' ', city) as 'address'
from customer where pcode like '9712%';
```

ID	address
2	Broerstraat 5 9712CP Groningen
3	OBS 34 9712GK Groningen

```
select cID * 2 as 'ID2',
       lower(street_addr) as street, upper(city) as 'city'
from customer where street_addr like '_________%';
```

ID2	street	city
4	broerstraat 5	GRONINGEN
8	nijenborgh 9	GRONINGEN

Sorting the output

customer

cID	street_addr	pcode	city	points
1	A-weg 30	9718CW	Groningen	1
2	Broerstraat 5	9712CP	Groningen	1
3	OBS 34	9712GK	Groningen	1
4	Nijenborgh 9	9747AG	Groningen	3

```
select * from customer order by pcode;
```

customer

cID	street_addr	pcode	city	points
2	Broerstraat 5	9712CP	Groningen	1
3	OBS 34	9712GK	Groningen	1
1	A-weg 30	9718CW	Groningen	1
4	Nijenborgh 9	9747AG	Groningen	3

```
select * from customer order by points desc, pcode asc;
```

customer

cID	street_addr	pcode	city	points
4	Nijenborgh 9	9747AG	Groningen	3
1	A-weg 30	9718CW	Groningen	1
2	Broerstraat 5	9712CP	Groningen	1
3	OBS 34	9712GK	Groningen	1

Aggregate functions

book

bID	pages	year
1	130	1995
2	544	1995
3	213	2005
4	210	2012

SQL provides functions that compute some simple statistics over columns.

- **count**: number of rows that match
  select count(∗) **from** book **where year** = 1995; ⇒ 2
- **sum**: sum of the matching elements
  select sum(pages) **from** book; ⇒ 1097
- **avg**: average of the matching elements
  select average(pages) **from** book; ⇒ 274.25
- **min**: minimum of the matching elements
  select min(**year**) **from** book; ⇒ 1995
- **max**: maximum of the matching elements
  select max(cID) **from** book; ⇒ 4

Grouping the results

book

bID	pages	year
1	130	1995
2	544	1995
3	213	2005
4	210	2012

```
select year, count(*) as book_count from book
group by year;
```

year	book_count
1995	2
2005	1
2012	1

Combining results of multiple queries

Results of multiple queries can be combined using set operations.

- Set union (∪): (query1) **union** (query2);.
- Set intersection (∩): (query1) **intersect** (query2);.
- Set difference (−): (query1) **except** (query2);

Note 1: MySQL does not support intersect and except.
Note 2: There are other ways of obtaining the same results.

Union: example

book

bID	pages	year
1	130	1995
2	544	1995
3	213	2005
4	210	2012

```
(select * from book where year = 1995)
union
(select * from book where bID >= 4');
```

bID	pages	year
1	130	1995
2	544	1995
4	210	2012

Alternative:

```
select *
from book
where year = 1995 or bID >= 5;
```


Intersect: example

book

bID	pages	year
1	130	1995
2	544	1995
3	213	2005
4	210	2012

```
(select * from book where year = 1995)
intersect
(select * from book where bID > 1);
```

bID	pages	year
2	544	1995

Alternative:

```
select *
from book
where year = 1995 and bID > 1;
```

Except: example

book

bID	pages	year
1	130	1995
2	544	1995
3	213	2005
4	210	2012

```
(select * from student where year < 2000)
except
(select * from student where pages > 500);
```

book

bID	pages	year
1	130	1995

Alternative:

```
select *
from book
where year < 2000 and pages < 500;
```


The 'in' clause

book

bID	pages	year
1	130	1995
2	544	1995
3	213	2005
4	210	2012

in can be used to check set inclusion.

```
select *
from book
where year in (1995, 2012);
```

bID	pages	year
1	130	1995
2	544	1995
4	210	2012

in becomes more useful with sub-queries:

```
select *
from book
where year in
(select year from book where pages < 500);
```


Subqueries in 'from'

customer

cID	street_addr	pcode	city	points
1	A-weg 30	9718CW	Groningen	1
2	Broerstraat 5	9712CP	Groningen	1
3	OBS 34	9712GK	Groningen	1
4	Nijenborgh 9	9747AG	Groningen	3

```
select *
from customer (select cID, city  from customer) as tmp
where tmp.cID < 3;
```

cID	city
1	Groningen
2	Groningen
3	Groningen
4	Groningen

It is an unnecessary complication here, but in more complex queries this feature may come handy.

Simple queries: summary so far

```
select attribute1, ···, attributeN
from table_name1, ···, table_nameM
where condition;
```

- **from** lists the table(s) for the query, **where** selects the rows, and **select** selects the columns.
- Result of an SQL query is a single (unnamed) table.
- In **where** clause you can use complex predicates. Strings have additional operations.
- You can rename the column names of the resulting table using **as**.
- SQL provides mechanisms to sort and do simple statistical calculations over the data.
- SQL allows set operations and nested queries.
- Note that there may be multiple ways of expressing the same query.

Queries on multiple tables

book

bID	pages	year
1	130	1995
2	544	1995
3	213	2005
4	210	2012

orders

cID	bID	qty
1	1	1
1	2	1
3	1	3
4	3	1

```
select book.bID, book.year, orders.cID,  orders.qty
from book, orders
where book.bID = orders.bID;
```

bID	year	cID	qty
1	1995	1	1
2	1995	1	1
1	1995	3	3
3	2005	4	1

Note: if you do not specify a **where** clause, you get the Cartesian product.

Table aliases

book

bID	pages	year
1	130	1995
2	544	1995
3	213	2005
4	210	2012

orders

cID	bID	qty
1	1	1
1	2	1
3	1	3
4	3	1

Sometimes it is clearer to give short aliases (temporary names) to the tables that are involved in the query

```
select b.bID, b.year, o.cID, o.qty
from book b, orders o
where b.bID = o.bID;
```

Naturally the result is the same:

bID	year	cID	qty
1	1995	1	1
2	1995	1	1
1	1995	3	3
3	2005	4	1

Natural join

book

bID	pages	year
1	130	1995
2	544	1995
3	213	2005
4	210	2012

orders

cID	bID	qty
1	1	1
1	2	1
3	1	3
4	3	1

The previous example was doing a natural join implicitly, we can get the same effect with **natural join** expression.

```
select bID, year, cID, qty
from book natural join orders;
```

Result is (again) the same

bID	year	cID	qty
1	1995	1	1
2	1995	1	1
1	1995	3	3
3	2005	4	1

You can join more than two tables using the same syntax:
t_1 **natural join** t_2 **natural join** t_3 ...

SQL basics: summary

- As well as being a query language, SQL is also a DDL and DML.
- DDL statements include **create table**, **alter table** and **drop table**.
- DML statements include **insert into**, **delete from** and **update**. Other commands can also manipulate data (such as **select** ... **into**).
- Basic form of SQL queries is:

```
select attribute1, ···, attributeN
from table_name1, ···, table_nameM
where condition;
```

What is next?

- More on joins.
- Null values.
- Indexes.
- Views.
- Access control.
- Reading for next week: Intermediate SQL (Chapter 4).
- Lab/Homework: (more) SQL exercises, posted online, due next Monday before the course.