Introduction to SQL and MySQL

Code: sql-intro

Author and version

- Daniel K. Schneider
- Email: Daniel.Schneider@unige.ch

- Version: 1.1 (modified 26/1/10 by DKS)
- Also available as wiki article: **http://edutechwiki.unige.ch/en/SQL_and_MySQL_tutorial**

Prerequisites: None

Objectives

- Learn basic SQL
- Learn how to use MySQL

Disclaimer

- There may be typos (sorry) and mistakes (sorry again). Please also consult a textbook !

Table of Contents

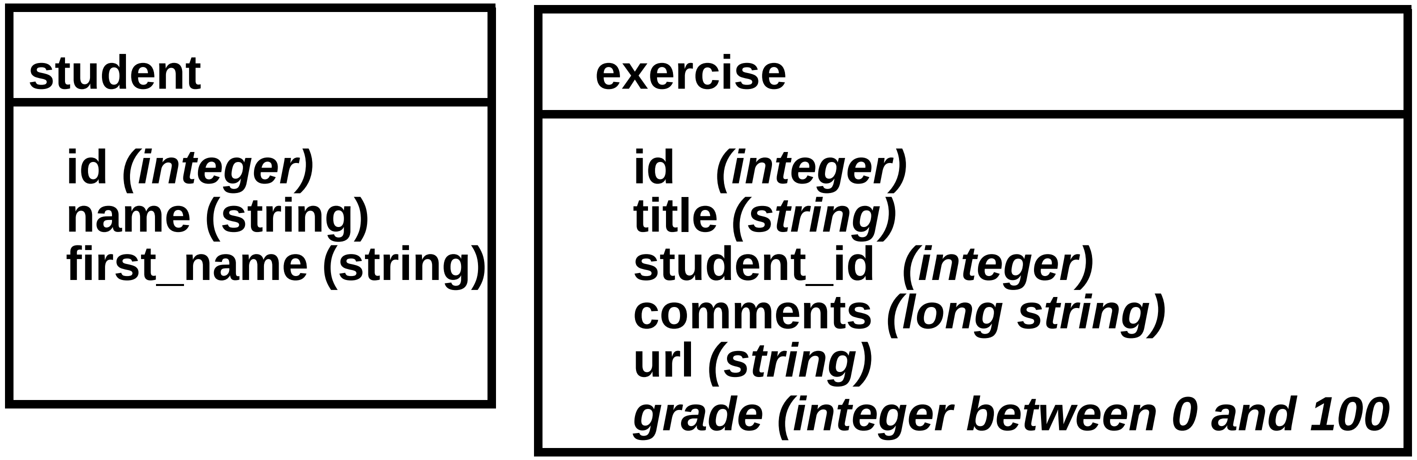
1. Some concepts about relational databases

1.1 Tables and relations

A relational database:

- contains one or more or more ***tables*** that contain ***records*** also called lines
- Each record is made of ***fields*** also called ***columns***.
- Each record (line) represent an ***information entity*** (an object described by attributes).
- Usually, the first field is used to insert a unique identifier for a record.
- Some tables include relations (i.e a column includes an identifier that corresponds to an identifier in an other table)

Example of two table structures: student with 3 fields and exercise with 6 fields



Fields describe different data types, e.g. integer numbers (int) or character strings (char).

- some fields are special keys that must be unique (more later)

Data are shown as tables, e.g. data we could have in the student table:

student table	**id**	**name**	**first_name**
	5	Miller	Joe
	2	St-Hypolite	Jean
	13	Müller	Paul

Information retrieval from a table

You need:

- Name of the table, e.g. "student"
- Names of columns and search criteria, e.g. name
- E.g. "I want all records from table exercise where grade > 60".

.... we shall see below how this is done with the SQL language

1.2 The SQL language

Most relational databases are implemented with SQL (Structured Query Language).

SQL is a sort of database programming language that allows you to

1. formulate queries, i.e. find stuff (SELECT)
2. manipulate records (UPDATE, INSERT, DELETE)
3. define tables and columns, as well as redefine and remove (CREATE, ALTER, DROP)
4. define access rights to database users (GRANT, REVOKE)

SQL syntax overviews. See:

url: **http://edutechwiki.unige.ch/en/SQL**

Example 1-1: Definition of simple table:

url: http://tecfa.unige.ch/guides/php/examples/mysql-demo/main.html

```
Database: demo  Table: demo1  Rows: 1
+----------+------------+------+-----+---------+----------------+
| Field    | Type       | Null | Key | Default | Extra          |
+----------+------------+------+-----+---------+----------------+
| id       | int(10)    |      | PRI | 0       | auto_increment |
| login    | varchar(10) |      | MUL |         |                |
| password | varchar(100)|  YES  |     |         |                |
| fullname | varchar(40) |      |     |         |                |
| url      | char(60)   |      |     |         |                |
| food     | int(11)    |      |     | 0       |                |
| work     | int(11)    |      |     | 0       |                |
| love     | int(11)    |      |     | 0       |                |
| leisure  | int(11)    |      |     | 0       |                |
| sports   | int(11)    |      |     | 0       |                |
+----------+------------+------+-----+---------+----------------+
```

- ignore details for the moment
- URL above is locked (ask the instructor for a password, login = coap). Alternatively, you can copy/paste the SQL code into your own database management system. See 3.5 “Table creation (CREATE)” [22].

2. Query (selection)

- SELECT allows to retrieve records from one or more tables

Here is a rough summary of its syntax:

```
SELECT [STRAIGHT_JOIN] [SQL_SMALL_RESULT] [DISTINCT | DISTINCTROW | ALL]
    select_expression,...
    [INTO OUTFILE ’file_name’ export_options]
    [FROM table_references
        [WHERE where_definition]
        [GROUP BY col_name,...]
        [HAVING where_definition]
        [ORDER BY {unsigned_integer | col_name} [ASC | DESC] ,...]
        [LIMIT [offset,] rows]
        [PROCEDURE procedure_name] ]
```

In this course we will work with simpler statements like:

```
SELECT select_expression1
FROM table_references WHERE where_definition2 ORDER BY col_name

(see next slides ...)
```

2.1 Simple SELECT

Syntax: **SELECT** *field1,field2,...* **FROM** *table*

Syntax: **SELECT * FROM** *table*

Example 2-1: Simple selections

- Retrieve fields (id,login,fullname,love,sports) for all records in table demo1.

```
SELECT id,login,fullname,love,sports FROM demo1
+----+-----------+------------------+------+--------+
| id | login     | fullname         | love | sports |
+----+-----------+------------------+------+--------+
|  1 | test      | Tester Test      |    3 |      3 |
| 34 | colin2    | Patrick Jermann2 |    1 |      4 |
....
```

- Retrieve all fields from table demo1.

```
SELECT * FROM demo1
+----+-----------+---------+------------------+------------------------+------+--.....
| id | login     | password| fullname         | url                    | food | w.....
+----+-----------+---------+------------------+------------------------+------+---.
|  1 | test      | 098cd4c | Tester Test      | http://tecfa.unige.ch  |    3 |    ...
| 34 | colin2    | b9hhhex | Patrick Jermann2 | http://tecfa.unige.ch/ |    1 |    ...
```

2.2 Conditional selection (SELECT .... WHERE)

- Instead of retrieving all records, you can add a filter, i.e. a "where clause"

  `Syntax:` **`SELECT .... FROM`** *`table`* **`WHERE`** *`condition`*

A. Overview table:

Operator	explanation
simple comparison operators	
=	equal
<> or !=	not equal
<	Less Than
>	Greater Than
<=	Less Than or Equal To
>=	Greater Than or Equal To
combination operators	
AND	both propositions need to be true
OR	one proposition needs to be true
special operators	
expr IN (..., ...)	value is in a list
expr NOT IN (..., ..., ...)	not in a list ....
expr BETWEEN min AND max	value is between
expr NOT BETWEEN ...	no in between ....
comparison operators for strings	

Operator	**explanation**
expr1 LIKE expr2	x is like y wildcards: % respresents several characters, _ represents one character
expr NOT LIKE expr2	not like...
expr REGEXP pattern	x is like (using regular expressions)
expr NOT REGEXP pattern	not like regular expression
STRCMP (exp1, exp2)	string comparison (C/PHP like).
Control flow	
IF (expr1, expr2, expr3)	If expr1 is true, return expr2, sinon expr3
IfNull (expr1, expr2)	Si expr1 est vraie, return expr1, sinon expr2
Mathematical functions	
see the manual ...	

Notes:

- Use parenthesis in a longer expressions to make sure to get what you want
- Strings should be included in ***straight*** quotes or double-quotes '...' or "..."

Examples:

- See next slides

Example 2-2: Simple Select ... where

- Retrieve parts of the records where love is bigger than 4

```
SELECT id,login,fullname,love,sports FROM demo1 WHERE love>4

+----+----------+------------------+------+--------+
| id | login    | fullname         | love | sports |
+----+----------+------------------+------+--------+
|  3 | colin    | Patrick Jermann  |    6 |      4 |
|  4 | schneide | Daniel Schneider |    6 |      6 |
+----+----------+----------------+-----+--------+
```

Example 2-3: Select ... where

```
SELECT * from demo1 WHERE login = 'colin' AND food < 6
```

Example 2-4: Select ... where ... IN

- Return the fullname of all records where login is either ’colin’ or ’blurp’

```
SELECT fullname from demo1 WHERE login in ('colin', 'blurp')
```

Example 2-5: Select ... where ... BETWEEN

```
SELECT * from demo1 WHERE food BETWEEN 3 AND 5
SELECT fullname from demo1 WHERE food BETWEEN 3 AND 5 AND love > 2
```

Example 2-6: Select ... where ... LIKE

- Find all records that include 'Patrick' in the fullname field. We use the LIKE clause with the % wildcard operator.

```
SELECT id,login,fullname,love,sports FROM demo1
        WHERE fullname LIKE '%Patrick%';
```

Result:

```
+----+----------+------------------+------+--------+
| id | login    | fullname         | love | sports |
+----+----------+------------------+------+--------+
|  3 | colin    | Patrick Jermann  |    6 |      4 |
| 93 | michelon | Michelon Patrick |    6 |      6 |
+----+----------+------------------+------+--------+
```

Example 2-7: Select ... where ... REGEXP

```
SELECT * from demo1 WHERE fullname REGEXP 'P.*J.*'
SELECT login,fullname from demo1 WHERE fullname REGEXP 'P.*J.*';
```

Result:

```
+--------+------------------+
| login  | fullname         |
+--------+------------------+
| colin2 | Patrick Jermann2 |
| blurp  | Patrick Jermann2 |
```

2.3 Result sorting (SELECT ... ORDER)

- Select all records (lines) and sort according to the id field

```
SELECT * from demo1 ORDER by id
```

- Same thing, but DESC = will sort in reverse order

```
SELECT * from demo1 ORDER by id DESC
```

2.4 Count records

- Count all lines (not null)

```
SELECT COUNT(*) FROM demo1
```

- Return counts of records having the same login

```
SELECT login, COUNT(*) FROM demo1 GROUP BY login;
```

2.5 Use of more than one table

- Fields are identified with the following syntax: `name_table.name_column`

Example 2-8: Select in 2 tables, see also • “BAD: ’ and ‘ and ” and “” [22]

```
SELECT demo1.fullname FROM demo1, test WHERE demo1.login = test.login
+-------------+
| fullname    |
+-------------+
| Tester Test |
```

3. Table definition

Creation of a table implies

- ***give the table a name***
- ***define fields (columns):*** type, size, default values, ...
- ***add other constraints to fields***
- ***grant permissions (sometimes)***

More or less complete syntax

```
Syntax: CREATE TABLE [IF NOT EXISTS] table_name
  (create_definition1,...) [table_options] [select_statement]
```

[1]create_definition:

```
col_name type [NOT NULL | NULL] [DEFAULT default_value] [AUTO_INCREMENT]
        [PRIMARY KEY] [reference_definition]
 or   PRIMARY KEY (index_col_name,...)
 or   KEY [index_name] KEY(index_col_name,...)
 or   INDEX [index_name] (index_col_name,...)
 or   UNIQUE [INDEX] [index_name] (index_col_name,...)
 or   [CONSTRAINT symbol] FOREIGN KEY index_name (index_col_name,...)
        [reference_definition]
 or   CHECK (expr)
```

... ignore for now, we will introduce the basics through the next slides...

3.1 Identifiers

A. General rules

- concerns: database names, tables, columns, etc.
- Keep the name below 30 characters
- Authorized characters: letters, numbers, #, $, _
- First character must be a letter
- Don’t use any accents, e.g. decision is ok, décision is not !
- Don’t use any SQL keywords, e.g. do not use SELECT, WHERE, etc.
- Note: SQL is not case sensitive ...

B. Tables and fields

- You may use the same field name in different tables
- Complete field (column) name:

```
Syntax: database.table.column
ex: demo.demo1.login
ex: demo1.login
```

3.2 Data types

Not all RDMS implement all data types, MySQL implements the most important ones.

Strings:

- delimiters: '....' or " ....."
- Special characters need to be quoted with \:
  \n (newline), \r (CR), \t = (tab), \’, \", \\, \%, _
- Quotes can be included within other quotes, e.g. ' "hello" ', " 'hello' " (no spaces)

Optional attributes (see next slides)

- UNSIGNED : only positive numbers
- ZEROFILL : inserts 0s, ex. 0004)

Optional parameters (see next slides)

- M used with integers: display size (not min/max values).
- (M,D) used with floating numbers: stored digits in total, digits after the "." (avoid if you don’t need this)
- M used with strings: length or max. length of a string

The NULL value

- Values can be NULL (means “empty”, not zero or empty string "" !!)

Data types summary table (some)

Type	explanation	range	example
NUMBERS			
TinyInt[(M)][UN-SIGNED] [ZEROFILL]	tiny integer	-128 à 127 (0 à 255)	TinyInt(2) 9
SmallInT[(M)]...	small integer	-32768 à 32767 (0 à 64K)	20001
MediumINT[(M)]...	integer	-8388608 to 8388607	-234567
INT[(M)] ...	integer	-2147483648 to 2147483647	
BigINT[(M)]...	big integer	63bits	
FLOAT(precision)	floating point		12.3
FLOAT[(M,D)]...	floating point	-3.402823466E+38 to -1.175494351E-38	float (5.2) 12.3
DOUBLE[(M,D)]...	big floating point		
DATES			
DATE	date	YYYY-MM-DD	3000-12-31
DateTime		YYYY-MM-DD HH:MM:SS	
TimeStamp[(M)]			
TIME			
YEAR			
STRINGS			

Char(M) [binary]	fixed-length string
	(avoid since comparisons are difficult ! Use varchar instead)
		M = 1 à 225 chars
		case insensitive (except binary)
			char(4)
			'ab '

VarChar(M)[binary]	variable length string	M = 1 à 225 chars	login(8)[binary] 'schneide'

Type	explanation	range	example
Texts and blobs			
BINARY(M)	fixed-length bit string (binary)		
VARBINARY(M)	variable-length binary		
TINYBLOB	small binary texts	255 chars	
BLOB		65535 chars	
MEDIUMBLOB		16777215 chars	
BLOB	big binary text	4294967295 chars	
TINYTEXT	small texts	255 chars	
TEXT		65535 chars	
MEDIUMTEXT		16777215 chars	
LONGTEXT	big text	4294967295 chars	
Enumeration			
Enum('val1', 'val2',...)	member of a list of strings or NULL	65535 distinct values	'toto'
Set('val1', 'val2', ...)	on or more strings	64 members	('toto', 'blurp')

- Binary and blobs vs. char and text: The first will story data in binary format. Char and text will store text with a given character encoding and you can edit these fields with a database tool.
- In most respects, you can regard a BLOB column as a VARBINARY column that can be as large as you like.
- Similarly, you can regard a TEXT column as a VARCHAR column.

Example 3-1: Creation of a simple table (CREATE)

```
CREATE TABLE pet (name VARCHAR(20), owner VARCHAR(20), species VARCHAR(20),
sex CHAR(1), birth DATE);
```

3.3 Keys

- Keys are critical. Speeds up data retrieval a lot !

A. Simple column keys (KEY)

- Indexed columns will improve database performance
- Each table can include 16 indexed columns
- All types (except blob and text) can be indexed, but must have non-NULL values !!
- Indexing of CHAR and VARCHAR can be reduced to first few characters

```
Syntax: KEY index_name (col_name)
Syntax: KEY index_name (char_col_name(M))
Note: INDEX synonymous of KEY
```

B. Primary KEY

- Primary keys uniquely identify a record (line)
- Therefore you can’t use a same value in more than one record, you cannot define a default value either...
- Most often, integers are used as primary keys
- Most often, primary keys are automatically generated

```
Syntax: PRIMARY KEY (index_col_name, index_col_name)
id int(10) NOT NULL auto_increment,
PRIMARY KEY (id),
```

3.4 Definition of fields

Note: See the complete example in section 3.5, p. 22

Example 3-2: Columns of demo1

```
id int(10) NOT NULL auto_increment,
login varchar(10) DEFAULT '' NOT NULL,
password varchar(100),
url varchar(60) DEFAULT '' NOT NULL,
food int(11) DEFAULT '0' NOT NULL,
```

Minimalist definition of a column:

`Syntax: name type`

Ex: id int

Some field types require a length definition, e.g. VarChar and Char !!

`Ex: login varchar(10)`

Typical definition of a column

`Syntax: name type (size) DEFAULT 'value_default' NOT NULL,`

Ex: login varchar (10) DEFAULT '' NOT NULL,

Definition of a primary key:

`Syntax: name type [size)] NOT NULL [auto_increment],`

Ex: `name int(10) NOT NULL auto_increment,`

- Keys are always defined with a separate statement, e.g.

```
PRIMARY KEY (id),
KEY login (login)
```

3.5 Table creation (CREATE)

```
Syntax: CREATE TABLE table (column1  spec1, column2 spec2, keys, )
```

Example 3-3: La table demo1

```
CREATE TABLE demo1 (
  id int(10) NOT NULL auto_increment,
  login varchar(10) DEFAULT '' NOT NULL,
  password varchar(100),
  fullname varchar(40) DEFAULT '' NOT NULL,
  url varchar(60) DEFAULT '' NOT NULL,
  food int(11) DEFAULT '0' NOT NULL,
  work int(11) DEFAULT '0' NOT NULL,
  love int(11) DEFAULT '0' NOT NULL,
  leisure int(11) DEFAULT '0' NOT NULL,
  sports int(11) DEFAULT '0' NOT NULL,
  PRIMARY KEY (id),
  KEY login (login)
);
```

- Make sure to separate each column or key definition with a comma
- End the whole CREATE statement with a ;
- Make sure that your quotes are straight !!!
  - GOOD: ' and "
  - BAD: ’ and ‘ and ” and “

4. Insertion and updates

4.1 Insert new records

INSERT allows to insert new lines (record) in one or more tables. There are two ways:

Example 4-1: INSERTION of a complete new line:

```
INSERT INTO demo1 VALUES (NULL,’colin’, ’b9hhhfa9347all893u483’, ’Patrick
Jermann’,’http://tecfa.unige.ch/’,1,2,1,3,4)
INSERT INTO demo1 VALUES
(5,’user12’,’098f6bcd4621d373cade4e832627b4f6’,’Testuser’,’www.mysql.com’,1
,4,5,2,1);
```

Example 4-2: INSERTION of a new line but specifying only a few values.

```
INSERT INTO demo1 (login, fullname, food) VALUES (’test2’, ’Patrick Test’,4)
```

- Attention: this can only work:
  - if a field is defined with default values (and not null)

    ```
    food int(11) DEFAULT ’0’ NOT NULL,
    ```
  - if a field is minimally defined. In this case NULL will be inserted (something you should avoid)

    ```
    fun int(11)
    ```

You will get an error

- if you try to enter a new id (primary key) that is already in the database
- if you don’t enter data for fields that require NOT NULL , but have no default value defined.

4.2 Updating

- UPDATE allows to update several fields for a selection (one or more lines !)

```
Syntax: UPDATE [LOW_PRIORITY] tbl_name SET
   col_name1=expr1,col_name2=expr2,...
   [WHERE where_definition]
```

Example 4-3: UPDATE examples

Update of the (’sports’) field for user (’michelon’):

```
UPDATE demo1 SET sports=3 WHERE login=’michelon’;
```

Update of two fields (’love’ et ’leisure’) :

```
UPDATE demo1 SET love=5, leisure=4 WHERE login=’michelon’;
```

Update with some math (add 3 to sports)

```
UPDATE demo1 SET sports=sports+3 WHERE login=’test2’
```

If you only want to update a precise record:

- Always use the "primary key" !!
- You can’t rely on names and such. In the above example ’michelon’ is a primary key....

4.3 Killing a record

Example 4-4: Kill lines

- To kill all lines (be careful !)

  ```
  DELETE FROM people;
  ```

- To kill a single line using a primary key:

  ```
  DELETE FROM people WHERE Id=1;
  ```

5. Relational tables

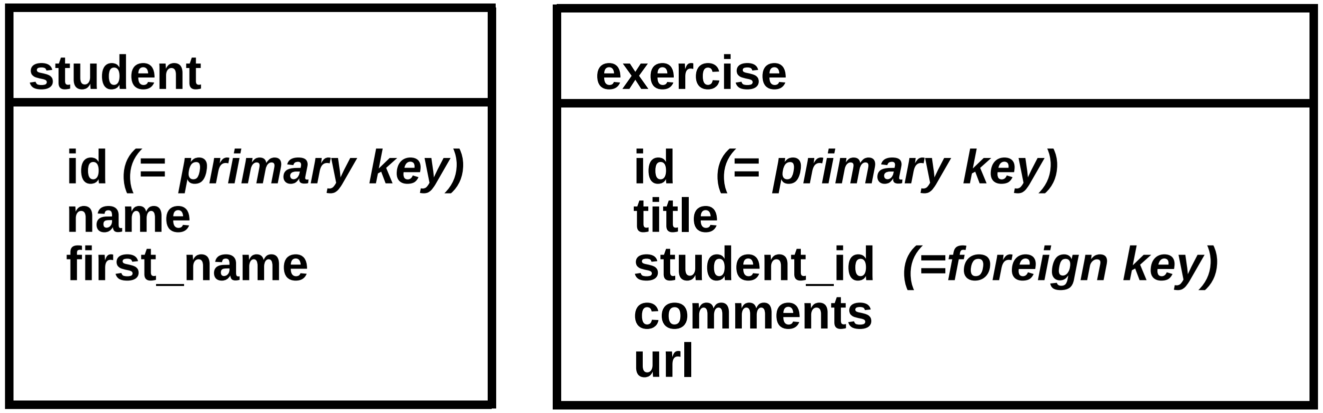
- This is a more difficult chapter. We just provide some basics here !
- Usually, databases contain several tables and that are related through use of keys. Each table represents an ***entity, (i.e. a thing)*** and its columns represent its attributes or properties.
- The tricky thing is to figure what is "entity" (i.e. a table) and what is a property (i.e. a table column or field). If properties can be shared with others, or if one of a kind can be multiple, then they should be modeled as a kind of entity, i.e. a new table. E.g. in a **`students`** table, one could add some columns that describe the school he attends, but since other students may attend the same school, one rather would create a new table describing **`schools`** using various properties. One then could just insert a number in the **`students`** table that represents the number of the school in the **`schools`** table.
- Remark: Serious database designers may model these entities and relationships before they start thinking about tables. Something that we will not really do here....

5.1 1-N relations

Frequently, relations are of type “1-to-N”. In this case one entity relates to 1 or more entities of a different kind.

Simple example:

- A simple application to register exercise grades for a group of students
- We use 2 tables: One to register students and the other one for the exercises
- Each student ("1") can turn in several exercises ("N")
- ***exercise.student_id*** corresponds to ***student.id***
- The primary key is on the “1” side ("id" in the "student" table)
- and it is inserted on the “N” as so-called foreign key. ("student_id" in the exercise table)



Example 5-1: File student_exercise.sql:

```
DROP TABLE IF EXISTS student;
DROP TABLE IF EXISTS exercise;

CREATE TABLE student (
  id int(10) NOT NULL auto_increment,
  name varchar(40) DEFAULT ’’ NOT NULL,
  first_name varchar(40) DEFAULT ’’ NOT NULL,
  PRIMARY KEY (id)
);

INSERT INTO student VALUES (NULL,’Testeur’,’Bill’);
INSERT INTO student VALUES (NULL,’Testeur’,’Joe’);
INSERT INTO student VALUES (NULL,’Testeuse’,’Sophie’);

CREATE TABLE exercise (
  id int(10) NOT NULL auto_increment,
  title varchar(40) DEFAULT ’’ NOT NULL,
  student_id int(10) NOT NULL,
  comments varchar(128),
  url varchar(60) DEFAULT ’’ NOT NULL,
  PRIMARY KEY (id),
  KEY student_id (student_id)
);
INSERT INTO exercise VALUES (NULL,"exercise 1",’1’,"pas de commentaire",’http://
tecfa.unige.ch/’);
INSERT INTO exercise VALUES (NULL,"exercise 2",’1’,"pas de commentaire",’http://
tecfa.unige.ch/’);
```

Playing with this example

- You can copy/paste instructions from the previous slide into phpMyAdmin
- Alternatively, use the command line interpreter (see 8.2 “Batch processing” [37])
  For example:

```
mysql -u schneide -p demo < student_exercise.mysql
```

Some queries:

- List exercises turned in for all students

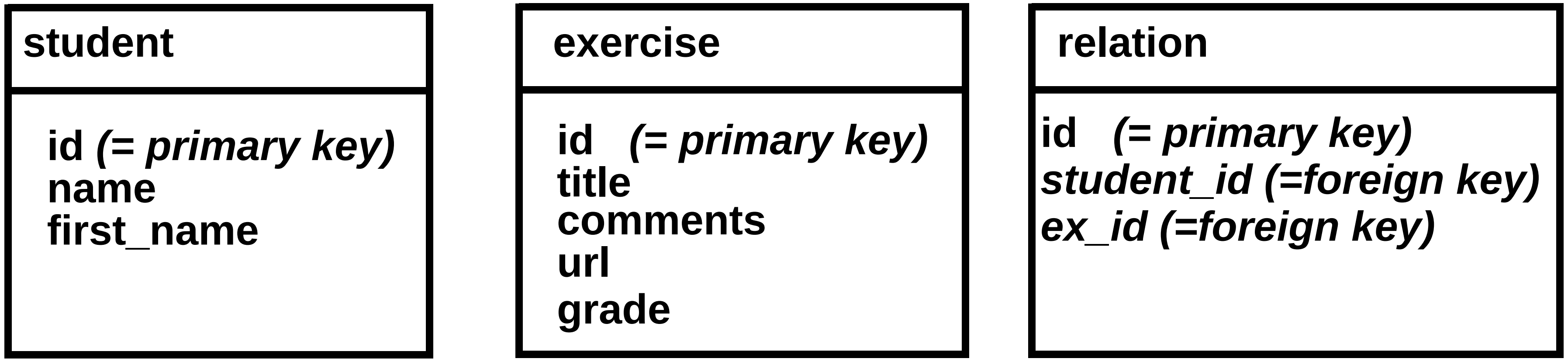
```
select * FROM student,exercise WHERE student.id = exercise.student_id;
```

- List only a few columns

```
select student.name, student.first_name, exercise.title, exercise.url FROM
student,exercise WHERE student.id = exercise.student_id;
+---------+------------+------------+------------------------+
| name    | first_name | title      | url                    |
+---------+------------+------------+------------------------+
| Testeur | Bill       | exercise 1 | http://tecfa.unige.ch/ |
| Testeur | Bill       | exercise 2 | http://tecfa.unige.ch/ |
+---------+------------+------------+------------------------+
```

5.2 N-N relations

- Let’s look at the same example. Some may argue that this model is not good enough, since some students could do exercises together. In that case we would need three tables: one that includes student information, one that defines exercises and one that relates the two:

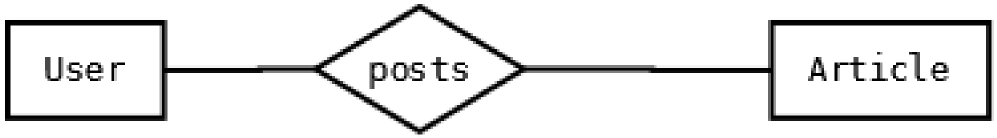


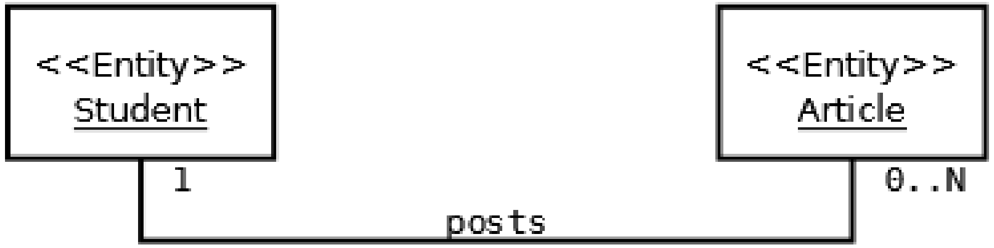
5.3 Modeling relations between entities

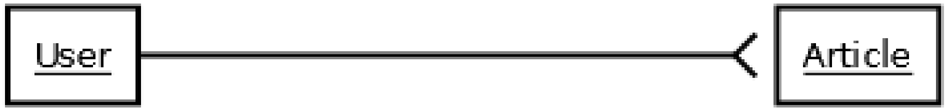
It may be a good idea to represent graphically relationships between entities before you start coding SQL. Before we show a few ways to draw these relationships, let’s quote from Wikipedia:

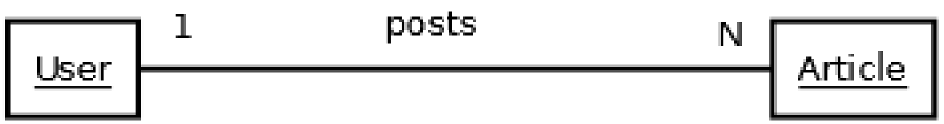
- An entity may be defined as a thing which is recognized as being capable of an independent existence and which can be uniquely identified. An entity is an abstraction from the complexities of some domain. When we speak of an entity we normally speak of some aspect of the real world which can be distinguished from other aspects of the real world. An entity may be a physical object such as a house or a car, an event such as a house sale or a car service, or a concept such as a customer transaction or order.
- Entities can be thought of as nouns. Examples: a computer, an employee, a song, a mathematical theorem. Entities are represented as rectangles.
- A relationship captures how two or more entities are related to one another. Relationships can be thought of as verbs, linking two or more nouns. Examples: an owns relationship between a company and a computer, a supervises relationship between an employee and a department, a performs relationship between an artist and a song, a proved relationship between a mathematician and a theorem. Relationships are represented as diamonds, connected by lines to each of the entities in the relationship.
- Entities and relationships can both have attributes. Examples: an employee entity might have a Social Security Number (SSN) attribute; the proved relationship may have a date attribute. Attributes are represented as ellipses connected to their owning entity sets by a line.
- Every entity must have a minimal set of uniquely identifying attributes, which is called the entity's primary key.

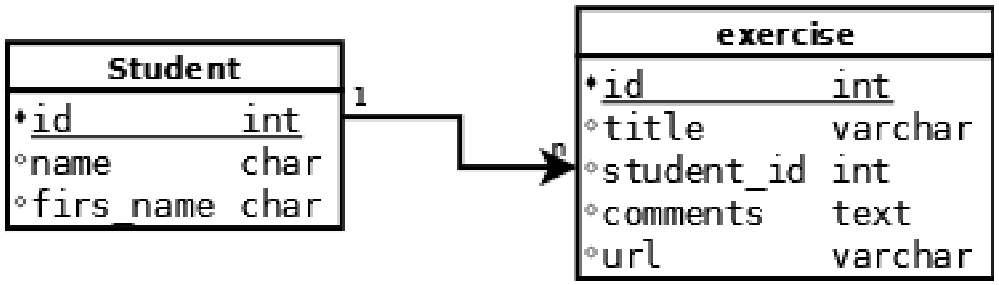
Example notations











6. Modification or deletion of a table

- Note: To do this, you need special administrators rights over the database or the table.

6.1 Destruction of a table

- Think, before you do this ....

```
Syntax: DROP TABLE [IF EXISTS] table
```

```
ex: DROP TABLE demo2
ex: DROP TABLE IF EXISTS demo2
```

6.2 Changing the structure of a table

- See the manual for details

```
Syntax: ALTER TABLE table ......
```

To add a column:

```
Syntax: ADD [COLUMN] create_definition [FIRST | AFTER column_name ]
```

```
ex: ALTER TABLE demo2 ADD COLUMN fun int(11) DEFAULT ’0’ NOT NULL AFTER love;
```

To kill a column:

```
Syntax: DROP [COLUMN] column_name
```

```
ex: ALTER TABLE demo2 DROP fun;
```

7. Permissions - Grants

- In an RDBMS you can assign different rights to different users for each database or even each table.
- In most context, it’s enough to define rights at database level (not at table level)
- Most often you assign these rights through the database administration interface.

Types of rights

- Read Data (SELECT)
- Write Data (INSERT, UPDATE, DELETE) records
- Structure Administration (CREATE, DROP, ALTER) of tables
- Database administration (GRANT, SUPER, RELOAD, SHUTDOWN etc ....)

  Typically, to install web applications a database user must have the first three types of rights, to use an application the first two are enough.

SQL statements

```
GRANT SELECT, UPDATE ON my_table TO some_user, another_user
REVOKE ....
```

- See manuals, as we said you usually do this through the admin interfaces...

8. Command line use of MySQL

- in case you like it “the old way” ....

8.1 Command line interface

- Remember that all SQL instructions must be separated by "`;`" (!!!)

A. Connection to a MySQL server

```
Syntax: mysql -u user -p [data_base]
```

Connection to a MySQL server on a different machine

```
Syntax: mysql -h host_machine -u user -p [data_base]
   -h: type the name of the server (if needed)
   -u: MySQL user (not the unix login !)
   -p: will prompt for a password
```

```
mysql -h tecfasun5 -u schneide -p
Enter password: ********
```

B. use/change of database (USE)

mysql> USE demo;

or alternatively:

mysql -u user -p demo

C. List tables (SHOW)

```
mysql> SHOW TABLES;
+----------------+
| Tables in demo |
+----------------+
| demo1          |
| test           |
```

D. Describe structure of a table (DESCRIBE)

```
mysql> DESCRIBE demo1;
+----------+------------+------+-----+---------+----------------+
| Field    | Type       | Null | Key | Default | Extra          |
+----------+------------+------+-----+---------+----------------+
| id       | int(10)    |      | PRI | 0       | auto_increment |
| login    | varchar(10) |     | MUL |         |                |
| password | varchar(100)| YES |     | NULL    |                |
| fullname | varchar(40) |     |     |         |                |
| url      | varchar(60) |     |     |         |                |
| food     | int(11)    |      |     | 0       |                |
| work     | int(11)    |      |     | 0       |                |
| love     | int(11)    |      |     | 0       |                |
| leisure  | int(11)    |      |     | 0       |                |
| sports   | int(11)    |      |     | 0       |                |
+----------+------------+------+-----+---------+----------------+
```

8.2 Batch processing

```
mysql -u user -p demo < test.sql
```

- Content of file test.sql is piped into SQL
  - see 3.5 “Table creation (CREATE)” [22]
- Don’t forget to include the name of the database (“demo” in the above example) !
- Note: if a table already exists you can’t define a new one with the same name. Kill it with “DROP TABLE if exists” (or use ALTER to change its structure)

  `Syntax: DROP TABLE [IF EXISTS]` *`table`*

  Ex: DROP TABLE demo2
  Ex: DROP TABLE if exists demo4

8.3 Backups

- If you “dump” a database you will create all the necessary SQL instruction to create it again (including all the INSERTs)

Use the ’mysqldump’ utility:

```
Ex: mysqldump -u schneide -p demo > save.mysql
```

8.4 List database, tables, etc.

... a few examples

Note: You may not be allowed to list all databases...

List all databases on the same machine or a server:

```
mysqlshow -u vivian -p
mysqlshow -h tecfa -u vivian -p
```

List tables of a database

```
mysqlshow -u vivian -p data_base_name
```

List table definition

```
mysqlshow -h tecfa -u vivian -p vivian test
```

9. MySQL with the phpMyAdmin application

- phpMyAdmin is the most popular web-based MySQL administration tool

9.1 Database selection

- Select your database from the pull-down menu to the left
  - Table names are shown below
  - The main window allows you to make changes in tables and also to execute general SQL queries.

9.2 Create tables with the online form

- There is a table creation tool
- However, we suggest to create tables with SQL instructions. This way you will have a trace.

9.3 Create tables from an SQL instructions file

- Click on the SQL (in the menu bar). Now you can either:
  - import a file with SQL instructions
  - Copy/paste SQL instructions

9.4 Other features

- You can
  - create and destroy tables (if you have appropriate user rights)
  - create, modify table definitions
  - view and edit tables (records)