20/07/2019

HackYourFuture
Databases - Lesson 3

Class 4

Borja Romero
Lessons 1,2,3

Gina Stavropoulou
Lessons 1,2

Geert Van Pamel
Lessons 2,3

Content

- Revision - Homework of last week
- ALTER TABLE / DROP TABLE
- VIEWS
- Data conversion
- Functions
- Transactions
- SQL Injection
- Other databases - NoSQL
- BASE
- Any questions?
- Homework
- Additional Material

Revision

What did we learn last time?

Restructuring a table

DDL maintenance

Example ALTER MODIFY

Sometimes changing the datatype of a column is not possible “sur place”
… because some data conversion or some logic needs to be done.

One solution: Create a temporary column.

```
ALTER TABLE alterex ADD ordernumChar CHAR(9);

UPDATE alterex SET ordernumChar = CONCAT(ordernum);

ALTER TABLE alterex DROP COLUMN ordernum;

ALTER TABLE alterex RENAME COLUMN ordernumChar TO ordernum;
```

Remark that the order of columns can/will change.

Not all databases allow renaming a column...

```
ALTER TABLE alterex RENAME COLUMN ordernumChar TO ordernum;
```

ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'to ordernum' at line 1

Then, as a fallback solution, you might need to recreate the table...

ALTER or CREATE/INSERT/DROP/RENAME

Some changes cannot be made online.
Or you might need so many changes, that it is more easy to recreate the table.

How would you do it?

You need to use following scheme:

- Recreate the new (temporary) table new_able
- Then INSERT + SELECT the data into the new table
- DROP the old table
- RENAME TABLE new_table TO old_table

Example migrating a table

Some changes cannot be made online.
Recreate the table when needed.

Tip: you can select and reorder columns by replacing * by an explicit list of columns...

```
CREATE TABLE alterex_new (ordernum CHAR(9));

INSERT INTO alterex_new SELECT * FROM alterex;

DROP TABLE alterex;

RENAME TABLE alterex_new TO alterex;
```

Drop/Copy a table

To delete a table you can simply do:

```
DROP TABLE table;
```

Pay attention: Dropping a table means that you also delete the data.

When needed make a copy before changing metadata.

```
CREATE TABLE table_new
SELECT * FROM table;
```

You could also rename the table (quick “online” restore when needed)

```
RENAME TABLE table_new TO table_todelete;
```

VIEWS

Are virtual tables (temporary)

```
CREATE VIEW … (…) AS
SELECT *
FROM … JOIN … ON …
WHERE …
```

- Views can be used anywhere a table is used.
- They do not hold data!
- Whenever the table changes the view changes too.
- Can show variants based on the same physical table
- If you want to delete it: `DROP VIEW view_name;`
- No ORDER BY (remember: mathematical SETS are not ordered)

VIEWS Simple example

```
CREATE VIEW clock (now) AS
SELECT current_timestamp FROM DUAL;

SELECT now FROM clock;
```

2019-07-15 21:30:52

```
DROP VIEW clock;
CREATE VIEW clock (now, date) as select current_timestamp,
STR_TO_DATE(current_timestamp, '%Y-%m-%d') FROM DUAL;

SELECT date FROM clock;
```

2019-07-15

VIEWS (Exercise time!)

Create a view best_films for the 5 best rated films :)

```
+------------------+--------+
| title            | rating |
+------------------+--------+
| Predator         |      9 |
| Fight Club       |      9 |
| Troy             |      8 |
| Kramer vs Kramer |      8 |
| Seven            |      8 |
+------------------+--------+
```

```
CREATE VIEW best_films AS
SELECT title, rating FROM
films ORDER BY rating DESC
LIMIT 5;
```

```
+-----------------+
| Tables_in_imdb  |
+-----------------+
| actors          |
| best_films      |
| films           |
| roles           |
+-----------------+
```

VIEWS - Share a table from another database

```
USE world;

CREATE VIEW actors AS SELECT * FROM imdb.actors;
```

```
+-----------------+
| Tables_in_world |
+-----------------+
| actors          |
| city            |
| country         |
| countrylanguage |
+-----------------+
```

Tip: when you would move the imdb.actors table in your world application you only need to update the view -- not the programs.

VIEWS - Advantages

- Simplifies your front-end application
- Views can be shared (application/s, sql interactive query)
- Simplifies application logic and maintenance
- Allows for changing the physical database without the need for changing your application; e.g. splitting a table in a JOINed table -> view definition changes but not the view columns… transparent to the application(s)
- Your table might contain more future columns than your view actually uses
- Your table could contain full history (start date/end date) while your application only need the current situation (example: product price list)
- The order of columns in your table might be different from your view (database maintenance; recreating columns)

VIEWS - More functionality

- Different applications might require variants of table (other data formats, other columns)
- For security reasons certain columns are reserved for e.g. the manager or the director, or human resources (employee salary)
  - Another view can be made for the department
  - A 3rd view can be used for the company
  - A 4rd view can contain public data
  - Certain departments may not see certain rows… additional WHERE
- Based upon a table, a “pivot table” can be created
- Technical columns can be hidden (user that created the row, the last user that updated the row, creation date, modification date -> can be implemented via DEFAULT and update triggers)

Joining a one-row table

DUAL is a generic one-row table, that can be used in SELECT or JOINs.

A one-row table can be joined without join condition, without the risk of generating spurious data.

No join condition => you cannot JOIN, so use a comma.

Think about the remark from lesson 2 where we talked about forgetting a join condition => generating non-existing data.

```
SELECT a.*, date FROM alterex a, clock;
```

Remark why the alias is necessary (repeating columns).

CONVERT and CAST

In the above views FROM DUAL is not mandatory (in MySQL).
MySQL allows a SELECT without FROM.

CONVERT or CAST can used to do explicit data type conversion.

```
SELECT CONVERT(current_timestamp, DATE);
```
2019-07-15

```
SELECT CONVERT(current_timestamp, TIME);
```
13:03:40

```
SELECT CAST('2014-02-28' AS DATE);
```
2014-02-28

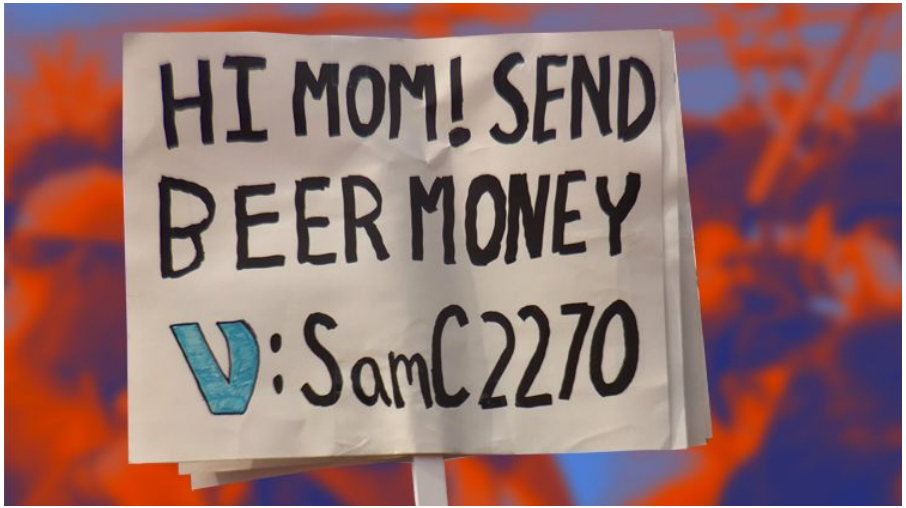
Transactions

CRUD - **C**reate **R**ead **U**pdate **D**elete

Why transactions?

- Users need to be sure data is correct
  - If you were a bank you would not allow money is “created”?
    - No credit without a debit...
- Multiple users may not influence each other
- User wants to have control over data updates
- In case of a (user) error, the user wants to undo a transaction
- Make auditing possible

Transaction



Mom -> 100 € - > Me

Which one you prefer?

```
update accounts
set money = money -100 where account_id = ‘MyMomId’;
update accounts set money = money + 100 where account_id = ‘MyId’;
```

VS

```
update accounts set money = money + 100 where account_id = ‘MyId’;
update accounts set money = money - 100 where account_id = ‘MyMomId’;
```

Transactions: Standard database behaviour

- A transaction is always generated (implicitly)
- A commit is automatically performed
- User cannot undo an automatic commit

Risks:

- User cannot undo an unwanted modification
- Unless he can execute himself the reverse operation
- What would you do to recover your error?

Transactions

```
SET autocommit = 0;      (implicit)

START TRANSACTION;       (explicit)

UPDATE …
INSERT …
DELETE …

COMMIT;                  (all is OK)
```

or

```
ROLLBACK;                (you made an error)
```

Transactions (Exercise time!)

Connect to imdb

Start a transaction

Update all the biographies to “awesome”.

Delete the row with id 9 from the roles table.

Insert a new film.

See the results on your tables.

You do not like the result -> reverse what you did :)

Commit

```
COMMIT;
```

Result:

- All DML transactions are permanently registered
- Caveat: DDL commands COMMIT pending transactions...

Rollback

```
ROLLBACK;
```

- All DML updates are undone
- Database returns at the state before the transaction started
- When there is an error, a rollback is/should be executed
- Or the error should first be corrected

Pay attention:

- If you do not COMMIT explicitly, the transaction is rolled back when exiting the client application (mysql)

DDL?

- Remember DDL cannot be rolled back and commits pending transaction
- DBA is a dangerous job
- Make a backup of your database (daily, before starting a great work)
- Think and verify before doing

Precaution: Explicit commit

With an explicit commit you still can make errors.

Once committed, the transaction cannot be undone by the user.

Protection via the DBA:

- Database backup
- Database or table restore (in time)

ACID

This is why we need transactions…

Atomicity: "all or nothing" - transactions, commit, rollback, journaling

Consistency: referential integrity - constraints, primary key, foreign key, triggers, cascade (delete)

Isolation: concurrent transactions do not see each other (readonly, read/write)

Durability: commit with caching + crash recovery (journaling)

Techniques:

- Locking (table, row) - deadlock
- Versioning, snapshotting (copy on write)

Security

Data access control

SQL injection

Major security problem for interactive (web scripting) programs (PHP, Perl, Python)

Use precompiled SQL statements and parameter binding instead of just interactive executing with straight parameter string concatenation...

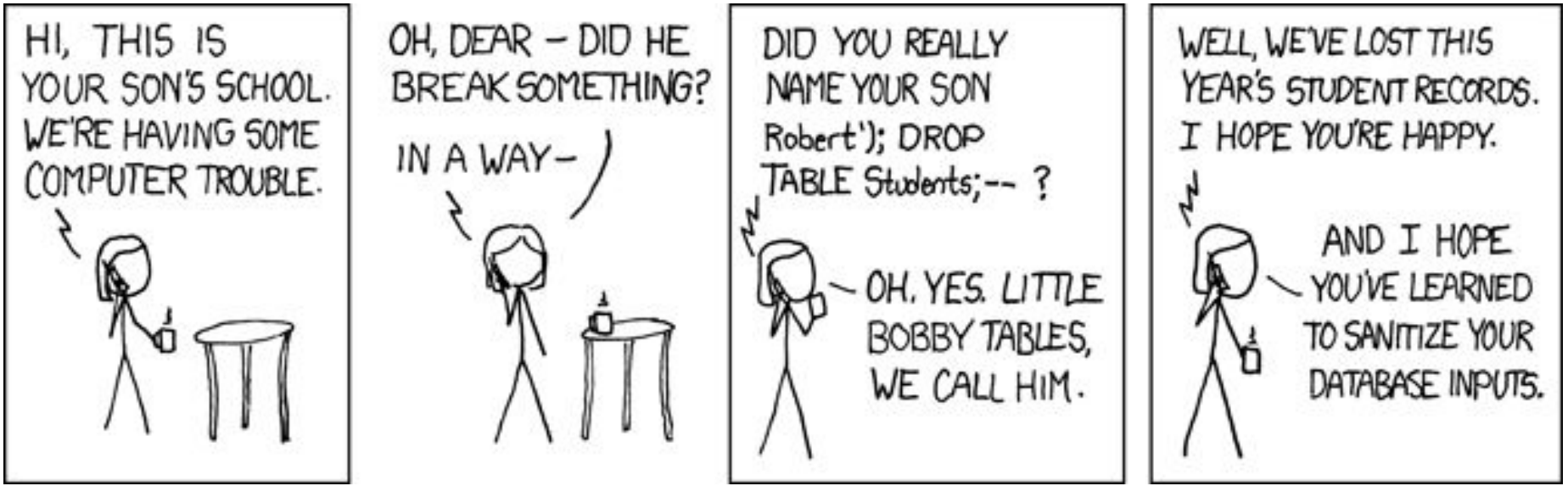
Bad code:

```
SELECT * FROM Employees where employeeID = (screen input);
```

What would happen if you enter: 1 or 1=1

Or with: 1; `DROP TABLE Employees;`

SQL injection



Security tips

- Have a separate development, testing, and production environment
- **Login with the least rights**
  - You can separate your data into multiple databases
    - One database per department?
  - Use multiple users with different rights (READONLY versus READ/WRITE)
  - Reference data is only READ by the application
    - The application only writes into transaction tables
    - Reference data is maintained by administrator (using separate user account)
    - Reporting is done on a replicated database (or by dedicated users)

- DDL is only needed for DBA
  - application only needs SELECT, UPDATE, DELETE

Database systems

Other systems

Relational databases

Strong points:

All relational databases are based on an ERD

- Logical data model
- Physical data model
- Internal representation

Order of columns is not important
Extensible due to relational data model

Transactions: ACID, multi-user.

Major examples

SQL is an ANSI language - implemented by:

- MySQL (what we have learned) - MariaDB (fork)
- Postgres (Geospatial database => allows queries with GPS coordinates)
- Oracle with PL/SQL (and reused technology from Rdb)
- Teradata (data warehousing)
- SQL Server (Microsoft Windows)
- (Microsoft Access)

In some databases you can program: stored procedures, functions, if then else, while, error handling, TCP/IP programming, calling external functions, batch processing.

Implementations

- Naming: Oracle names a ‘database’ a ‘schema’
- Long table names may not be supported by some databases (Teradata)
  Tip: make sure to make table name unique with 30 characters
- All database systems implement an *extended subset* of the ANSI standard
  Tips:
  - to remain compatible, try to not use extensions
  - if you insist (because it is easy) use the extension in a view
- Some databases might implement data types differently
  - Dates are an example of how databases can vary

Different behaviour

Some commands might not exist in other databases. Some commands might behave differently. Commands may be implemented differently.

- MySQL is case sensitive for database and table names
- MySQL is not case sensitive for column names
- MySQL by default is not case sensitive for string values

```
INSERT INTO alterex (ordernumChar) VALUES ('A123');
```

- Oracle is case sensitive for both columns and string values:

```
SELECT * FROM alterex WHERE ordernumChar = 'a123';
```

This statement will not find the order in Oracle.

MySQL storage engines

Tables can be of different implementations, each having other characteristics and functionality.

- MyISAM (old, more performant, no transactions => no concurrency, readonly)
- InnoDB (new, allowing transactions)
- Many other storage formats

Convert via mysqldump (export, import).

To see the available storage engines:

```
SHOW STORAGE ENGINES;
```

NoSQL

Other databases have been developed:
Keep it simple, dynamic, store everything, large volumes, distributed computing, internet… big data (servers, systems, clusters, nodes)

- XML database
- MongoDB
- In-memory databases
- Embedded database
- RDF database (triplestore)

Using different storage techniques, query tools.
Depending on the application you choose for a Relational db or another type of database...

Wikidata - Wikibase

- Triplestore Wikidata is an example of RDF/NoSQL:

Extremely simple physical data model: **One single table with only 3 columns**

Logical data model is implicit (based on properties, and soft relationships)
Not consistent, not complete, work in progress - can store anything
No transactions, no locks, no (real) constraints - but: full history is kept

Statements:

Subject - Property - Object (or value)

Query language: SPARQL - Application: MediaWiki - Implements: RDF

Wikidata web application

- Application: https://www.wikidata.org (running Wikibase)
- Query tool: https://query.wikidata.org (running SPARQL)
- Create a user account to update data
  - User account is shared with applications like Wikipedia (SUL - Single User Login)

- Examples:
  - https://www.wikidata.org/wiki/Q61976480 (HackYourFuture)
  - https://tools.wmflabs.org/reasonator/?q=Q61976480 (HackYourFuture)

- Documentation: https://www.wikidata.org/wiki/Wikidata:In_one_page
  - https://commons.wikimedia.org/wiki/File:Wikidata-in-brief-1.0.pdf
  - https://commons.wikimedia.org/wiki/File:Wikidata_Query_Service_in_Brief.pdf

BASE

Inverse of ACID…

Basically **A**vailable: You normally always get an answer (maybe outdated)

Soft state: No hard constraints/commits - you always can insert data

Eventual consistency: Collaborative content

Users make sense of data; not a pre-designed data model.

See also: CAP theorem

Node.js web application with database integration

User client	Pug HTML code generator	Database backend
Web Browser	Application logic	Tables/views
	Database connectivity	Access control

Any questions?

This was only an introduction on databases.

We did not talk about:

- Autojoin (join a table with itself; recursion, trees, hierarchies)
- Triggers (insert, update, delete)
- User defined functions
- Database administrator activities
- Physical tuning of your database
  - Indexes
- How to secure your database
- How to backup
- How to upgrade
- Development tools
- Database migration
- BLOBs
- CLOBs
- Storage
- Geospatial queries
- Reporting
  - PIVOT
- Data warehouse functionality

Homework

Exercises on Views

1. Redefine the clock view to add additional columns: add time, as INT: year, month, day of month, day of week, day of year, isWorkDay, dayNumber
   a. you can use the UNIX_TIMESTAMP() function…
   b. You can use CONVERT, CAST, or any other available functions
2. Create a view to redefine the order of columns in one of your tables
3. Create a view on the actors table to see BirthDate instead of age
   a. Recreate the actors table and replace age by birthDate
   b. Now you might create a view to see the age instead of BirthDate
4. Create a view that will give you information about the actors in this format:

   actors.fname actors.lname plays roles.rname in films.title

   ie: Brad Pitt plays Tayler in Fight Club

Exercises on Views (2)

1. Implement tracking of (reference) table updates
    a. Rename one of your data tables into _HIST
    b. Add a column start date and end date
    c. Use default start date as 1 January 1900 or 15 October 1582 (why?)
    d. Use default end date as 31 December 2100 or 31 December 9999 (why?)
    e. Create a view to return only the “current” rows (use BETWEEN)
    f. Explain why you could store future product price changes using this technique?

Optional: Exercises on Views (3)

1. Imagine that you need to design an employee table. How would you do this, given that you also need a (view) for manager, director, executives, and external personnel. This is kind of object oriented design.
    a. Implement optional (not applicable) columns
    b. Make an overview of mandatory and optional columns for each type of employee
    c. Think about security

Additional background info

Duplicate key constraint

One actor cannot play twice the same role (unless in 2 different films…?)

The following query can search for duplicate data.

```
SELECT aid, rname, COUNT(*) countRoles
FROM roles
GROUP BY aid, rname
HAVING countRoles > 1;

Empty set (0.00 sec)
```

Remember that you can use INSERT … SELECT which can fail with duplicate key. With the above command you can query the source table to find duplicates.

Rely on the system

Making it yourself easy

Query optimisation

- You can write the same logic in multiple ways
- The database engine is choosing the best algorithm (normally)
- … sometimes the database is choosing the wrong algorithm
- Good database design is critical for performance
- Sometimes the physical implementation is different from the logical model
- WHERE clauses normally require an additional index (to avoid sequential read)
- (for small tables sequential read is not bad)
- Non-referenced tables are omitted in JOINs

Physical optimisation

- NULL does not allocate storage (1 bit)
- You could create future columns without actually using them
- DEFAULT NULL does not make sense (it is standard behaviour)
- Column compression (frequently used values)
  - You could also use a separate JOIN table with IDs and descriptive text columns
    - Or use CHAR(x) codes instead of status text columns
- Block compression

Coding conventions

About Case:

In this course we teach you to type keywords like SELECT FROM WHERE in uppercase.

In real live you might type “select from where” in lowercase, because what really is important is the TABLENAME and the COLUMNAMES; not the keywords.

So you might reverse the convention.

Reading too much uppercase is difficult for a human being.

Coding conventions (2)

About command structure: the following might be better readable than writing everything on one single line -- for the machine it does not matter…

```
SELECT column 1
    , column 2
    , …
FROM …
WHERE …
AND …
HAVING …
ORDER BY …
```

Spaces in code can be important

The following command are logically correct, but generate syntax error…

Normally spaces in code are ignored, except:

- MySQL does not allow for a space between a function and “(“

```
SELECT CAST ('2014-02-28' AS DATE);
```

ERROR 1584 (42000): Incorrect parameters in the call to stored function `CAST`

```
SELECT TRIM ('a  ')
```

ERROR 1630 (42000): FUNCTION world.trim does not exist. Check the 'Function Name Parsing and Resolution' section in the Reference Manual

About (trailing) spaces and string padding

CHAR and VARCHAR could behave differently.

Comparing with = could behave differently.

GUI

Instead of the command line, there exist also GUIs to manipulate databases.

- Local clients
- Web clients

Very handy. You should use one.

But here we want you to first learn the commands…
… so we were using just the mysql command prompt. (Sorry about that but it is the best way to learn)

Database lifetime

A database is like a living being or a machine:

- Your application evolves
- The software version of the database evolves
- It changes (content, structure, functionality)
- It requires maintenance (indexes, backup, restore)
- It can become sick or broken (corrupt, file deleted) => backup
- Security problems
- Disk full error
- Data can get lost (wrong delete, data manipulation error)

Further advise

Do not reinvent the wheel:

- Less can be more: KISS
- The system does it better
- Use what the DBMS has available
- But: isolate the system dependencies from the application (use views)
- Because all databases can be different
- Reuse what others already have developed
- Ask Google search
- Learn from others
- Ask others

DUAL table

1. Some database systems would not have a DUAL table.
    a. What would you do in that case?
    b. Create an example

Optional: generic data modelling

1. Instead of physical data modelling you might need to do generic data modelling. Generic data models are generalizations of conventional data models. They aggregate or virtualise relationships.
   Wikidata is an extreme form of generic data modelling.

   Design a group of tables that describes the relationship between people. One table should be the relationship_type table.

   E.g. Person X is married with person Y
   Person X is child of person Y
   Person X is father of person Y, etc.

Thank you…

See you later...