BITAH05 - Databanktechnologie

Jasper Anckaert

Overview

The student is able to
- Solve exercises on both normalisation and database creation
- Create a database model
- Recognise several database types
- Use online databases

Course material
- Slides on LEHO

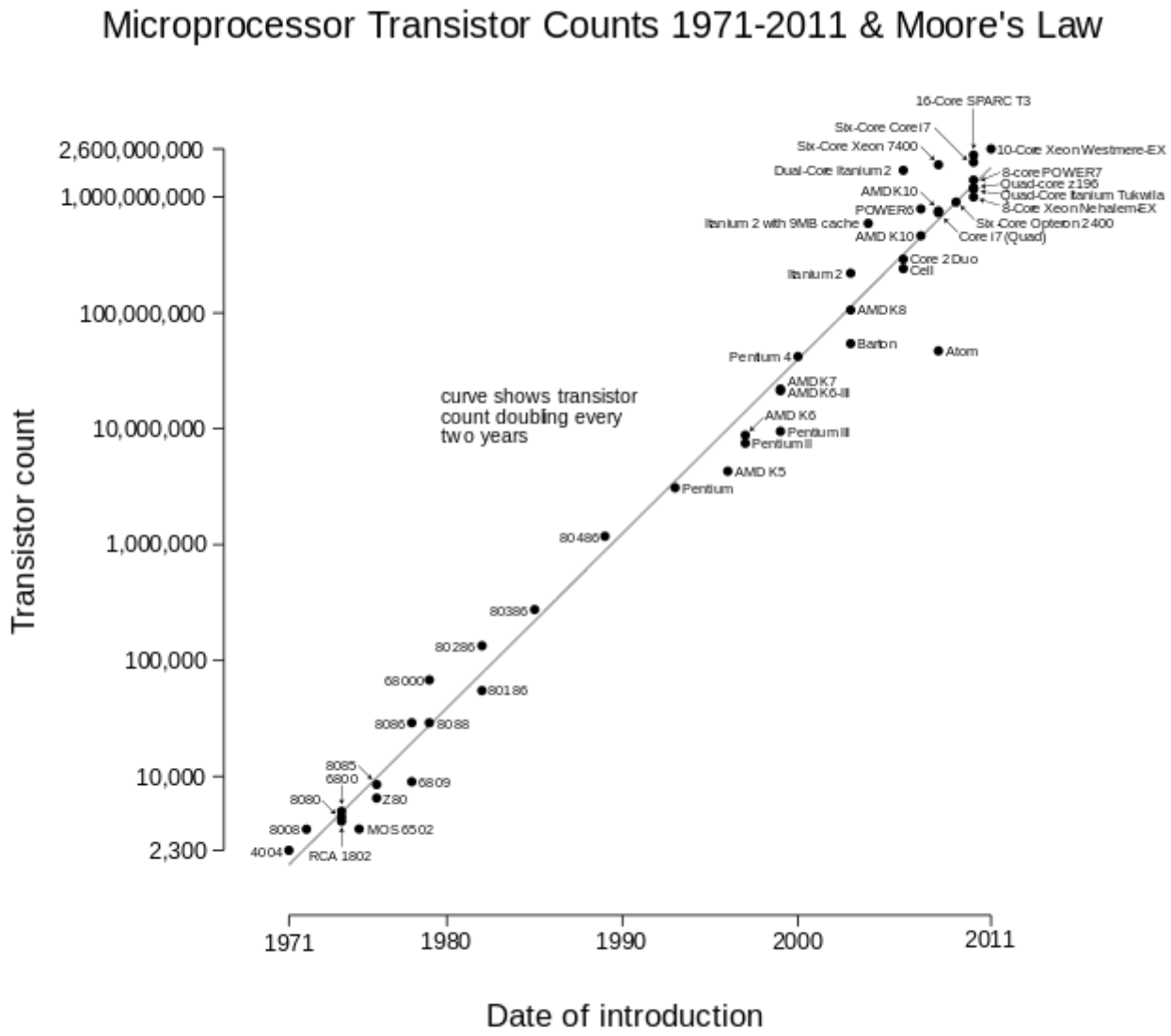
Examination
- Theoretical and practical part

Requirements

- A working internet connection

Lecture 1 – Introduction & relational databases

Introduction



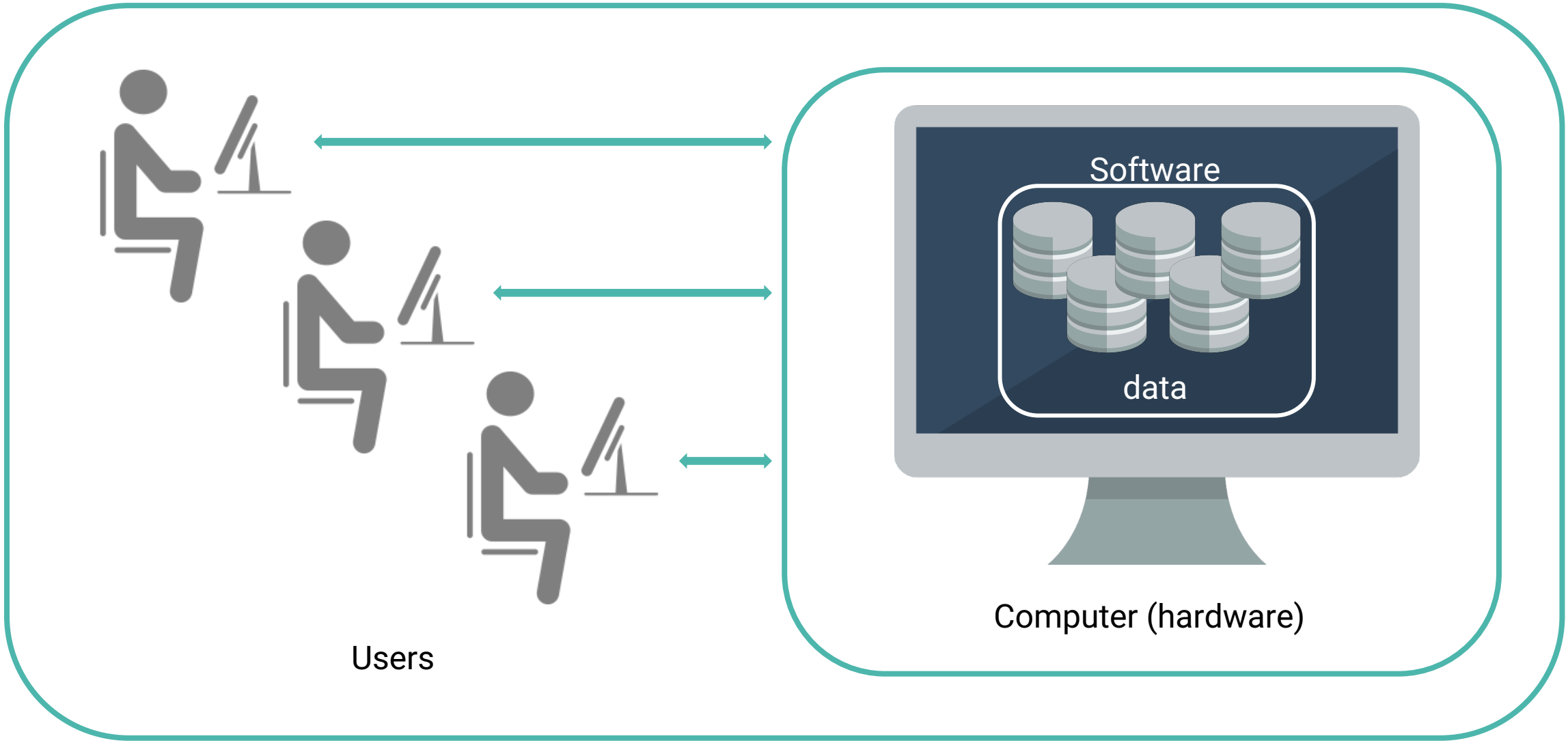
Introduction

Database

- Mechanism used to store information
- Collection of data (numbers, dates, text, …)
- Structured storage of data
- Efficient interaction with data
    - CRUD

- **Used everywhere!**

Introduction

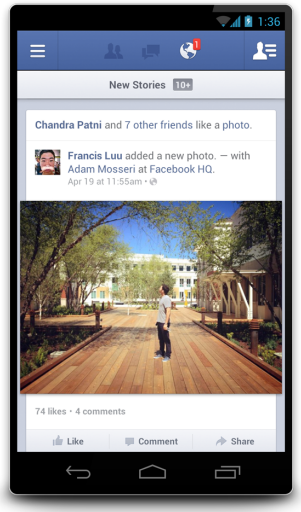


Introduction

Database system

- Hardware
  - Processor and internal memory
  - Personal laptop – server cluster
- Data
  - Structured storage
  - Little redundancy
  - Shared (single-user vs. multi-user)
- Software
  - Database Management System (DBMS)
    - Data storage
    - Data retrieval
    - Data manipulation
- Users
  - Authentication & authorization
  - Administrator > end-user

Introduction



View level

View 1

View 2

...

View N

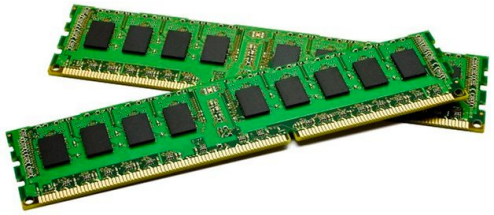
Patients
Patient_id INT PK
Patient_name
Demographic_Info INT FK
Indexes
PRIMARY id

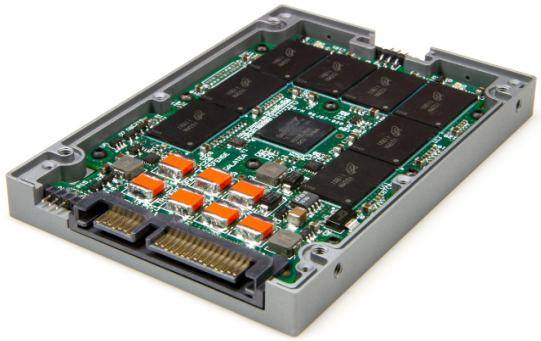
Medication
Medication_id INT PK
Patient_id INT FK
Medication_name
Indexes
PRIMARY id

Diagnoses
Diagnosis_id INT PK
Patient_id INT FK
Diagnosis
Indexes
PRIMARY id

Logical level

Logical data independence





Physical level

Physical data independence

Introduction

A good database system

- Sufficient amount of storage
  - Think ahead
- Easily accessible
  - Centralised system
- Secured against non-users
  - Per user rights
- Easily maintainable
  - Add, edit, delete data
- Controlled input
  - All data in same format
- Quick!
  - Queries need to be run within a certain time
- Little redundancy
  - Do not store the same information twice (or more)
  - Application dependable
- Clear structure

Introduction

Database Management System (DBMS)

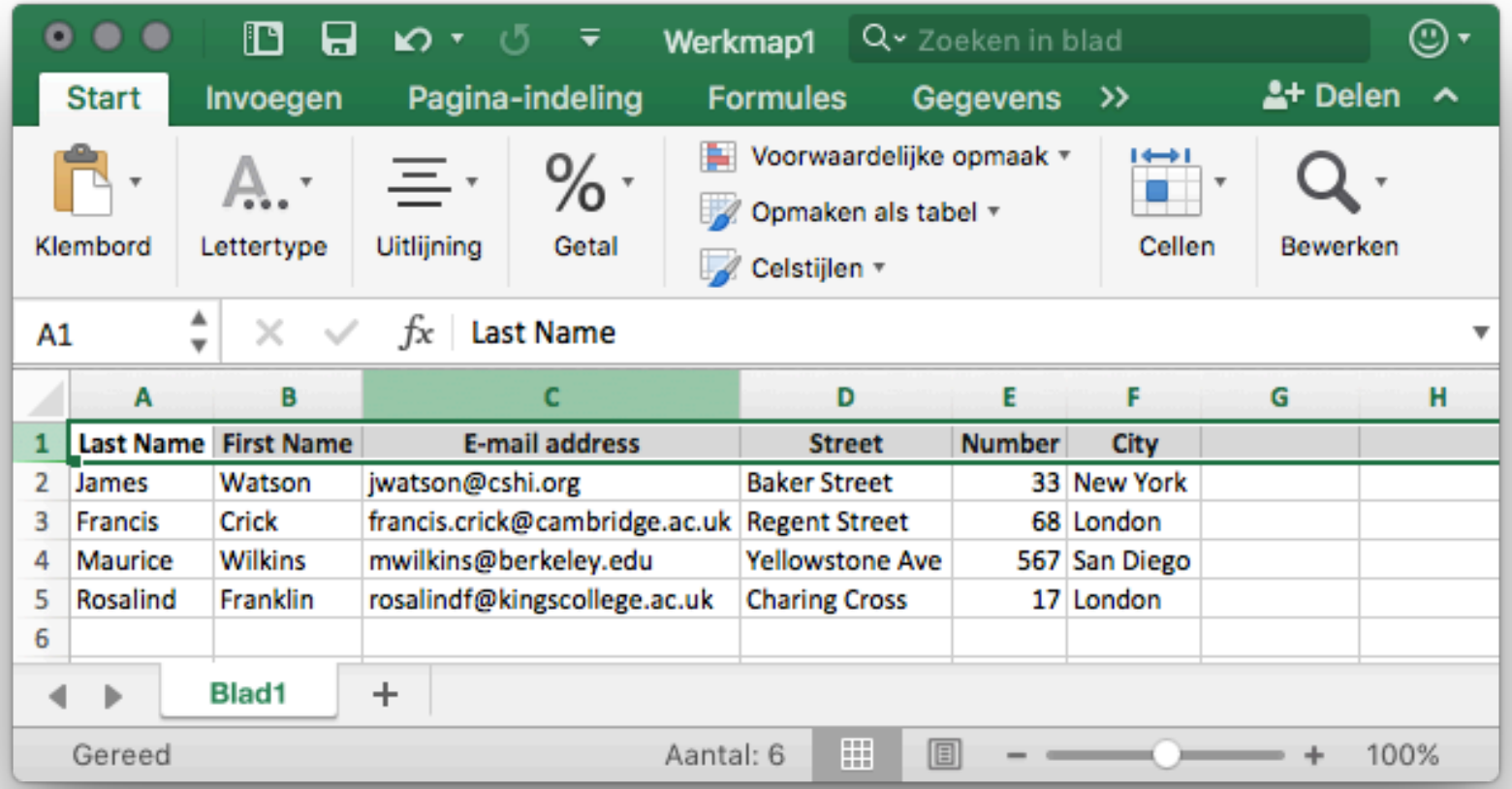
- A **database management system** (**DBMS**) is a computer software application that interacts with the user, other applications, and the database itself to capture and analyse data
- Shields users from hardware and storage details
- Most important software component, though not the only one (development tools, …)

- Used for
  - Data storage
  - Data retrieval
  - Data manipulation
  - Authentication & authorization

Introduction

Relational databases

- Rigid structure
- 2 dimensional *tables*
  - Columns (fields)
  - Rows (records)



	A	B	C	D	E	F	G	H
1	Last Name	First Name	E-mail address	Street	Number	City		
2	James	Watson	jwatson@cshi.org	Baker Street	33	New York		
3	Francis	Crick	francis.crick@cambridge.ac.uk	Regent Street	68	London		
4	Maurice	Wilkins	mwilkins@berkeley.edu	Yellowstone Ave	567	San Diego		
5	Rosalind	Franklin	rosalindf@kingscollege.ac.uk	Charing Cross	17	London		
6								

Introduction

Relational databases

- Model objects (entities) and their relationships
- E.g. *a store sells products to customers*
  - Entities:
    - Customers
      Attributes: name, address, telephone number, …
    - Products
      Attributes: name, price, …
  - Relationships:
    - Sale
      Attributes: quantity, timestamp, …

Introduction

Relational Database Management Systems (RDBMS)

- Enforce data integrity
  - Honours constraints on columns
- Enforce referential integrity
  - Honours constraints on relations

! 12 rules of Edgar Codd!

Introduction

Relational Database Management Systems (RDBMS)

- Commercial products
  - Oracle
  - DB2 (IBM)
  - MS SQL Server (Microsoft)
- Open-source
  - **MySQL (Oracle)**
  - PostgreSQL
  - SQLite

Introduction

Relational Database with MySQL

- Most used RDBMS
- Open source
- Free of charge (paying versions exist)
- Wordpress, Twitter, Facebook, …

- http://www.mysql.com

Introduction

Installing MySQL on your system

- Linux
  - http://dev.mysql.com/doc/refman/5.7/en/linux-installation.html
- Mac
  - http://dev.mysql.com/doc/refman/5.7/en/osx-installation.html
- Windows
  - http://dev.mysql.com/doc/refman/5.7/en/windows-installation.html

Introduction

Starting/stopping/restarting MySQL

- Linux/Mac

```
$ /etc/init.d/mysqld start    service mysqld start      service mysql start
$ /etc/init.d/mysqld stop     service mysqld stop       service mysql stop
$ /etc/init.d/mysqld restart  service mysqld restart    service mysql restart
```

- Windows

```
$ C:\> "C:\Program Files\MySQL\MySQL Server 5.7\bin\mysqld”
$ C:\> "C:\Program Files\MySQL\MySQL Server 5.7\bin\mysqladmin" -u root shutdown
```

Introduction

Check whether or not MySQL is running correctly

- Linux/Mac

```
$ service mysqld status       (service mysql status)
mysql start/running, process 3394
$ ps -ef | grep mysql
mysql 3394 1 0 12:09 ? 00:00:00 /usr/sbin/mysqld
$ netstat -ltpn | grep mysql
tcp 0 0 0.0.0.0:3306 0.0.0.0:* LISTEN 3394/mysqld
```

- Windows

```
$ C:\> "C:\Program Files\MySQL\MySQL Server 5.7\bin\mysqlshow"
$ C:\> "C:\Program Files\MySQL\MySQL Server 5.7\bin\mysqlshow" -u root mysql
$ C:\> "C:\Program Files\MySQL\MySQL Server 5.7\bin\mysqladmin" version status proc
$ C:\> "C:\Program Files\MySQL\MySQL Server 5.7\bin\mysql" test
```

  Or check running services through *Control Panel*

Introduction

The MySQL monitor

- To connect or log on to a MySQL database service

  ```
  $ mysql
  ```

- Many options, check the manual

  ```
  $ man mysql
  ```

  or

  ```
  $ mysql --help
  ```

Introduction

The MySQL monitor

- Most important options

```
$ mysql [options] [database]
    -u uname | --user=name
            default: UNIX account
    -p [pwd] | --password[=pwd]
            default: <none>
    -h hname | --host=name
            default: localhost
    -P prt | --port=prt
            default: 3306
```

Introduction

Exercises

- Connect to the database and execute the following commands

```
mysql> select current_user;
mysql> show databases;
```

Introduction

Securing the server

```
$ mysql_secure_installation
```

- Set password for root accounts
- Remove anonymous-user accounts
- Remove remote root login
- Remove test database

Introduction

Securing the server – extra

- Prevent any external access to the database server
  - Add to global config file (*/etc/mysql/my.cnf*)

```
[mysqld]
bind-address = 127.0.0.1
```

Introduction

Database users

- Database users and OS users are completely independent from each other
  - No user specified --> OS user is taken
  - Database superadmin --> *root@localhost* (all rights, including dropping databases)
- Not a good idea to always connect as root!!!

Introduction

Database user - privileges

- Created user has very limited privileges
- Grant privileges *prv* on table *tbl* in database *db*

```
mysql> GRANT prv ON db.tbl TO user@host;
```

- Some wild cards
  - All privileges, specify `all` as *prv*
  - All databases, specify * as *db*
  - All tables, specify * as *tbl*
- The given database and table names do not have to exist (yet)

Introduction

The options file

- Avoid retyping of password with every connection, create options file
  - `.my.cnf`
  - Located in home directory
  - Protect from others: mode 600
- Contains *key=value* pairs in [sections]
  - Provided as (invisible) command line parameters

Introduction

The options file - example

- Put password in options file
  - Command line parameters of mysql

    $ mysql --password=*pwd*
- Options file could look like this

```
[client]
password=pwd
user=username
```

Introduction

SQL: Structured Query Language

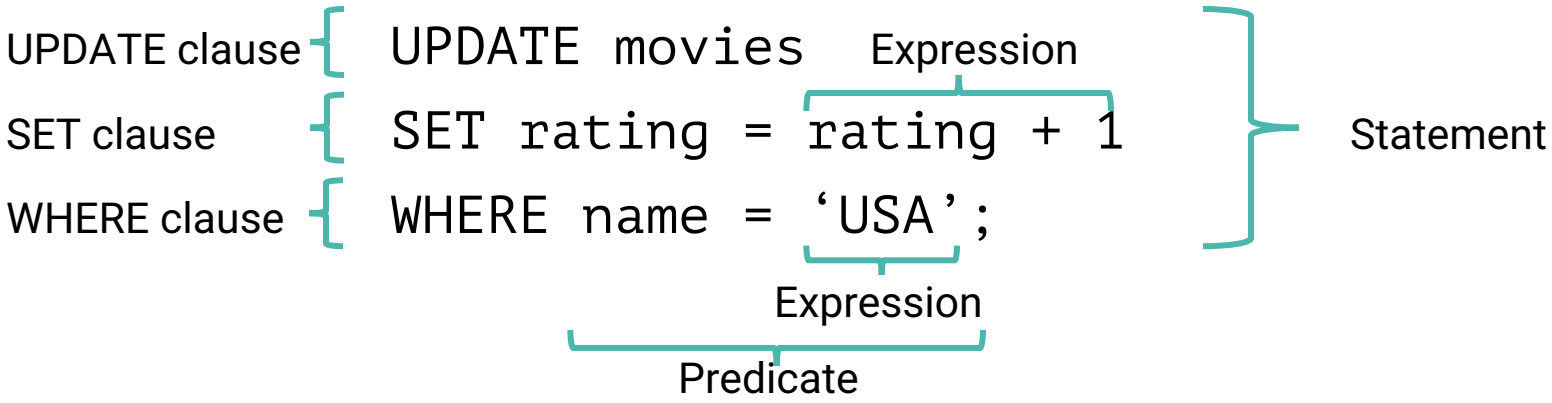
- **Data Definition Language** (DDL) statements: design (create, alter, drop, …) database
  `CREATE TABLE, DROP DATABASE`
- **Data Manipulation Language** (DML) statements: manage data
  - **C**reate: add new data
  - **R**ead: collect data
  - **U**pdate: alter data
  - **D**elete: remove data

  `SELECT, INSERT, UPDATE, DELETE`
- **Data Control Language** (DCL) statements: manage database rights
  `GRANT, REVOKE`

- **Transaction Control Language** (TCL) statements: manage DML tasks (group, undo, …)

Introduction

SQL: Structured Query Language



Introduction

The MySQL monitor (again)

- Several ways to execute SQL statements using the MySQL monitor
  - Interactively

    ```
    $ mysql [database]
    mysql> stmt
    ```

  - From the command line

    ```
    $ mysql [database] -e 'stmt'
    ```

  - From a file or a pipe (stdin)

    ```
    $ mysql [database] < stmt_file
    $ cat stmt_file | mysql [database]
    ```

Introduction

Creating a database

- Only database users with significant privileges can create databases
  - From the command line

    `$ mysqladmin [opt] create` *`dbname`*

    `mysqladmin` has the same command line options as `mysql`
  - From within the mysql monitor

    `mysql> create database` *`dbname`*

Introduction

Exercises

- As *root@localhost*, create database '*`biodb`*'
- Grant all privileges to the database user you created before
- Download the `1.sql` (first take a look at the contents)
- Execute all SQL statements in the file

Introduction

Hierarchy

- A single MySQL service can have multiple databases

  ```
  mysql > SHOW databases;
  ```

- A particular database *db* can have multiple tables

  ```
  mysql > USE db;
  mysql > SHOW tables;
  ```

- A particular table *tbl* can have multiple columns or fields

  ```
  mysql > SHOW columns FROM tbl;
  mysql > SHOW create table tbl;
  ```

Introduction

Exercises

- Connect to the database service as a normal user
- What databases do you see?
- What tables are defined in *biodb*?
- What are the column names?

Relational databases with MySQL

- MySQL database = collection of tables
  - Table = set columns with specific types
    - Number, text, date, …
    - Each row in same format
    - Only 1 value per field

Student_number	Name	Last_name	Birthdate	Sex
0293826	John	Doe	1991-10-02	M
0293749	Mel	Trotter	1991-04-11	V
0328273	Bill	Schuette	1990-12-01	M

Relational databases with MySQL

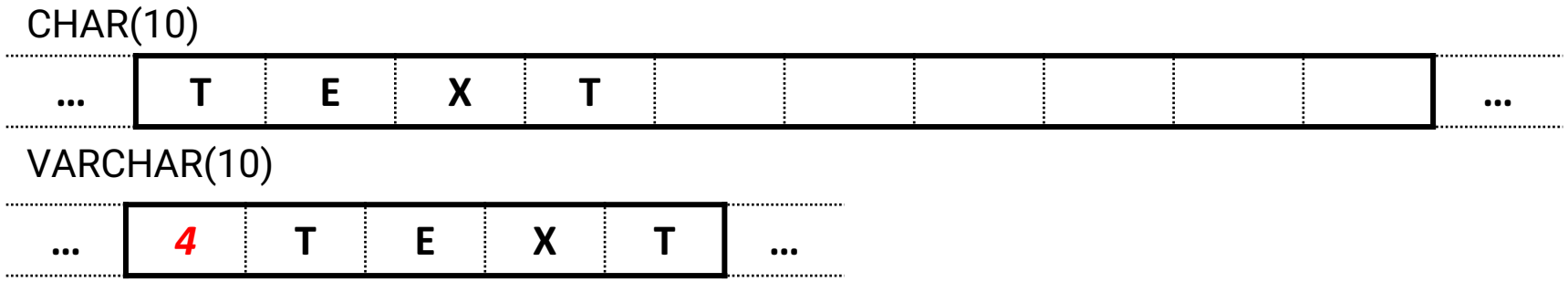
Column types

- INT
  - Integer
  - SIGNED: -2 147 483 648 tot 2 147 483 647
  - UNSIGNED: 0 tot 4 294 967 295
  - TINYINT, BIGINT, SMALLINT
- FLOAT & DOUBLE
  - Numbers with decimal point
  - FLOAT: 7 digits after decimal point, DOUBLE: 15 digits after decimal point
- DATE
  - YYYY-MM-DD
  - DATETIME
    - YYYY-MM-DD HH:MM:SS
    - ! TIMESTAMP ! No dates < 1970 and > 2038

Relational databases with MySQL

Column types

- VARCHAR & CHAR
  - String with a certain number of characters
  - Define max number of characters e.g. VARCHAR(200)
  - VARCHAR: up to 65 535 characters
  - CHAR: up to 255 characters, spaces are added to reach required length



  - VARCHAR is more efficient in storage, CHAR is faster for reading data
  - Similar for INT vs BIGINT vs ...

Relational databases with MySQL

Column types

- TEXT & BLOB
  - Used for texts that are not queried often or do not have to be searchable
  - BLOB for binary data (images, …)
- ENUM
  - List of permitted values
    - E.g. Set of colours: ’red’, ‘green’, ‘blue’
  - Very efficient

Relational databases with MySQL

Column types

Student_number	Name	Last_name	Birthdate	Sex
0293826	John	Doe	1991-10-02	M
0293749	Mel	Trotter	1991-04-11	V
0328273	Bill	Schuette	1990-12-01	M

Relational databases with MySQL

Column types

- Every row has to be unique
  - DBMS is able to distinguish separate rows

Student_number	Name	Last_name	Birthdate	Sex
0293826	John	Doe	1991-10-02	M
0293749	Mel	Trotter	1991-04-11	V
0328273	Bill	Schuette	1990-12-01	M

Primary key: column that makes sure every row is unique!

- Usually first column
- INT
- auto_increment: adds 1 to each value automatically

Relational databases with MySQL

Constraints

On top of column types, there are some additional requirements per column

- Primary key
  - Only 1 PK per table, all values must be unique
- UNIQUE
  - All values (or combinations) must be unique
- NOT NULL
  - Field can not be empty when adding data (empty = null)
- Default
  - Default value for a field
- Foreign key
  - Same constraints as referenced column
  - Security when adjusting linked data possible

Relational databases with MySQL

Summary – database structure

MySQL database

→ table

→ row

→ column

column type

constraint(s)

Relational databases with MySQL

`INSERT` – add new rows

```
INSERT INTO tbl (col1, col2) VALUES (val1, val2);
INSERT INTO students (student_number, name, last_name) VALUES
(2654897, ‘Glenn’, ‘Walker’);
```

- Be aware!
  - Not all columns need to be included in query, unmentioned columns are given the default value for that column
  - Empty column gets null value
  - Columns with NOT NULL constraint require a value
  - Strings are written between quotes

Relational databases with MySQL

`SELECT`– retrieve rows

```
SELECT columns FROM tbl;
SELECT * FROM modorg;
SELECT genus, species FROM modorg;
```

- *`columns`*
  - List of columns, separated by comma
  - * for all columns
  - Use of arithmic operators (+, -, …) and other functions (min, max, …) on columns is possible
- *`tbl`*
  - Single table or multiple tables joined together
  - Subquery
  - View

Relational databases with MySQL

`ORDER BY` – sort rows

```
SELECT columns FROM tbl ORDER BY col1 [asc|desc] [, col2
[asc|desc]...];
```

When using `SELECT` statements, the data is displayed in no particular order

→ use `ORDER BY` clause

- *colX*: a column or a column alias
- *asc*: ascending order (default)
- *desc*: descending order

Relational databases with MySQL

Exercices

- Show the names (genus & species) of all model organisms in the order of the publishing date of the draft
- Show the names (genus & species) of all model organisms sorted by the number of chromosomes (most chromosomes on top) and than alphabetically by name

Relational databases with MySQL

Calculated rows

- You can add columns in a query that calculate some value using other columns of the same row
- Lot of functions and operators readily available

```
mysql> SELECT 6*7;
mysql> SELECT concat(class, “ “, genus) FROM modorg;
mysql> SELECT now();
```

Relational databases with MySQL

Calculated rows – numbers

- Operators

  `+, -, *, /, %`

- Functions

  ```
  sqrt(x), power(x, y), …
  exp(x), ln(x), …
  sin(x), cos(x)
  round(x), ceil(x), floor(x), …
  rand(), rand(x)
  ```

Relational databases with MySQL

Calculated rows – strings

- Functions

```
length(s)
concat(s1, …)
upper(s), lower(s)
trim(s), ltrim(s), rtrim(s)
substr(s, …)
reverse(s)
truncate(s)
```

Relational databases with MySQL

Calculated rows – dates

- Functions

```
currentdate(), now()
year(d), month(d), week(d)
dayofmonth(d), dayofweek(d)
hour(d), minute(d), second(d)
```

Relational databases with MySQL

Exercices

- Show
  - Model organism full name (as one column)
  - Genome size (as Gb), rounded to 4 digits
  - Average chromosome size
  - Publication year

  of all rows sorted by average chromosome size (largest on top)

Relational databases with MySQL

Column aliases

- Columns can be renamed
  `SELECT` *`col`* `[AS]` *`alias`* …
- The aliases can be used in the `ORDER BY` clause

Relational databases with MySQL

Exercices

- Show
  - Model organism full name (as one column) as name
  - Average chromosome size as avgsize
  - Publication year as pubyear

  of all rows sorted by avgsize (largest on top)

Relational databases with MySQL

`WHERE` – filter rows

```
SELECT columns FROM tbl WHERE condition(s) [ORDER BY sortcol];
```

- *conditions*
  - One or more conditions, combined with `AND`, `OR`, `NOT`, `XOR`
  - Only row for which the `condition(s)` evaluates `TRUE` are selected
  - Unable to use column aliases in `condition(s)`

Relational databases with MySQL

Filtering rows – conditions

- Numerical comparison operators
  - `=`
  - `!=` or `<>`
  - `<`, `<=`, `>`, `>=`
  - `between` *x* `and` *y* (inclusive)

- E.g. select all organisms with more than 10 chromosomes

```
SELECT genus, species FROM modorg WHERE nchr > 10;
```

Relational databases with MySQL

Filtering rows – conditions

- String comparison operators
  - =
  - `!=` or `<>`
  - `<`, `<=`, `>`, `>=` (lexical)
  - `like “pattern”`
    matches a pattern
    - `_` (A single character)
    - % (zero or more characters)
  - `rlike “regex”` [MySQL]
    matches a regular expression

- E.g. select all mammals

```
SELECT genus, species FROM modorg WHERE class = “mammals”;
```

Relational databases with MySQL

Filtering rows – conditions

- Dealing with NULL-values
  - Testing for NULL-ness

    `SELECT … WHERE col IS NULL;`

    `SELECT … WHERE col IS NOT NULL;`
  - Substitution of NULL-values

    `SELECT ifnull(col, value) …`
    this function returns
    - *col* if *col* is NOT NULL
    - *value* if *col* is NULL

    e.g. `SELECT genus, species, ifnull(nchr, 0) FROM modorg;`

Relational databases with MySQL

Filtering rows – conditions

- Boolean logic
  - *not x*
    Evaluates TRUE if *x* is FALSE
  - *x and y*
    Evaluates TRUE if both *x* and *y* are TRUE
  - *x or y*
    Evaluates TRUE if *x* or *y* is TRUE, or both
  - *x xor y (exclusive or)*
    Evaluates TRUE if *x* or *y* is TRUE, but not both

Relational databases with MySQL

Exercises

- Select all mammals with genomes published after 2005
- Select all organisms that have an average chromosome size between 10 and 100 Mbp
- Select all organisms whose genus starts with A, B, C, D, or E

Relational databases with MySQL

Filtering rows – duplicates

- Eliminate duplicate rows
  `SELECT DISTINCT(`*`cols`*`) FROM …`
  → Each combination of *`cols`* is unique

Relational databases with MySQL

Filtering rows – limiting output

- Limit the number of rows in a result set
  `SELECT … LIMIT` *n* `[OFFSET` *r*`];`

- Result set is limited to a maximum of *n* rows
- If an offset *r* is given, the first *r* rows are skipped
- Mostly used in combination with `ORDER BY`

Relational databases with MySQL

Exercises

- Give an overview of all organism classes in the dataset (sorted alphabetically)
- Show the organism names of the top 3 largest genome sizes

Relational databases with MySQL

Aggregation

- Queries are concentrated on particular rows
- Possible to calculate a single result across multiple rows
  e.g. maximum genome size?
- SQL allows you to
  - Specify criteria to group rows together
  - Calculate a single value per group
  - Filter grouped data

Relational databases with MySQL

Aggregation

- Functions
  - `count(col)`, `count(*)`, `count(distinct col)`
  - `sum(col)`
  - `min(col)`, `max(col)`
  - `avg(col)`, `stddev(col)`, `variance(col)`

Relational databases with MySQL

Exercises

- All these queries return a row count. What is the result and why?

```
SELECT count(*) FROM modorg;
SELECT count(nchr) FROM modorg;
SELECT count(class) FROM modorg;
SELECT count(DISTINCT class) FROM modorg;
```

- How many mammals are in the database?

Relational databases with MySQL

`GROUP BY` - aggregation

- Sort data into groups for aggregation purposes

  ```
  SELECT [col,] aggregatefunctions FROM src [WHERE cond] GROUP BY
  col [ORDER BY …];
  ```

- All rows with the same value in *`col`* are grouped
- For each group, the aggregate function is calculated
- No sense in select other columns than *`col`*

Relational databases with MySQL

Exercises

- How many organisms are present in the dataset for each class?
  Note the sort order.
- Show the minimum and the maximum genome sizes for each class.
  Take only those organisms into account for which genome sizes are known.
  Sort the results such that the biggest maximum genome size is on top.

Relational databases with MySQL

Aggregation – filtering

- Filter results based on the results of aggregate functions using HAVING clause

```
SELECT [col, ] aggregatefunctions FROM src [WHERE cond1] GROUP BY col HAVING cond2 [ORDER BY …];
```

- Column aliases can be used in *cond2*

Relational databases with MySQL

```
SELECT
    [ALL | DISTINCT | DISTINCTROW ]
      [HIGH_PRIORITY]
      [STRAIGHT_JOIN]
      [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
      [SQL_CACHE | SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]
    select_expr [, select_expr ...]
    [FROM table_references
    [WHERE where_condition]
    [GROUP BY {col_name | expr | position}
      [ASC | DESC], ... [WITH ROLLUP]]
    [HAVING where_condition]
    [ORDER BY {col_name | expr | position}
      [ASC | DESC], ...]
    [LIMIT {[offset,] row_count | row_count OFFSET offset}]
    [PROCEDURE procedure_name(argument_list)]
    [INTO OUTFILE 'file_name'
        [CHARACTER SET charset_name]
        export_options
      | INTO DUMPFILE 'file_name'
      | INTO var_name [, var_name]]
    [FOR UPDATE | LOCK IN SHARE MODE]]
```

http://dev.mysql.com/doc/refman/5.5/en/select.html

Relational databases with MySQL

Execution order

1. Input columns are determined
2. `WHERE` – input columns are filtered
3. `GROUP BY` – sorting & grouping of filtered input
4. Aggregation functions are calculated
5. `HAVING` – aggregation functions are filtered
6. `ORDER BY` – output is sorted
7. `LIMIT/OFFSET` – output is chopped

Relational databases with MySQL

Exercises

- For each class with more than 1 organism, show the average number of chromosomes. Sort the result such that the biggest average is on top.

Relational databases with MySQL

Database upgrade

- Download `2.sql`
- Create a new database `bioinf`

```
mysql> CREATE database bioinf;
```

- Create the tables and insert the data

```
$ mysql bioinf < 2.sql
```

Relational databases with MySQL

Joins

- Relation databases model entities and their relationships
- Different entities: different tables
- Allow you to combine information across different tables

Relational databases with MySQL

Joins

- What if we want to expand our database with a trajectory and course info

Student_number	Name	Last_name	Birthdate	Trajectory	Course
0293826	John	Doe	1991-10-02	FBT	Databases
0293749	Mel	Trotter	1991-04-11	MLT	Databases
0328273	Bill	Schuette	1990-12-01	MLT	Databases
0293826	John	Doe	1991-10-02	FBT	Scripting
0293826	John	Doe	1991-10-02	FBT	Linux

- Problem: redundant information
  - Waste of space
  - Error prone

Relational databases with MySQL

Joins

- Solution: relational databases with a **foreign key**

Students

Student_number	Name	Last_name	Birthdate	Trajectory_ID	Course
0293826	John	Doe	1991-10-02	1	Databases
0293749	Mel	Trotter	1991-04-11	2	Databases
0328273	Bill	Schuette	1990-12-01	2	Databases
0293826	John	Doe	1991-10-02	1	Scripting
0293826	John	Doe	1991-10-02	1	Linux

- Important: Data types of linked columns have to be equal! Foreign key is typically primary key of other table

Trajectories

ID	Trajectory
1	FBT
2	MLT

Relational databases with MySQL

Joins

Students

Student_number	Name	Last_name	Birthdate	Trajectory_ID
0293826	John	Doe	1991-10-02	1
0293749	Mel	Trotter	1991-04-11	2
0328273	Bill	Schuette	1990-12-01	2

Course_of_student

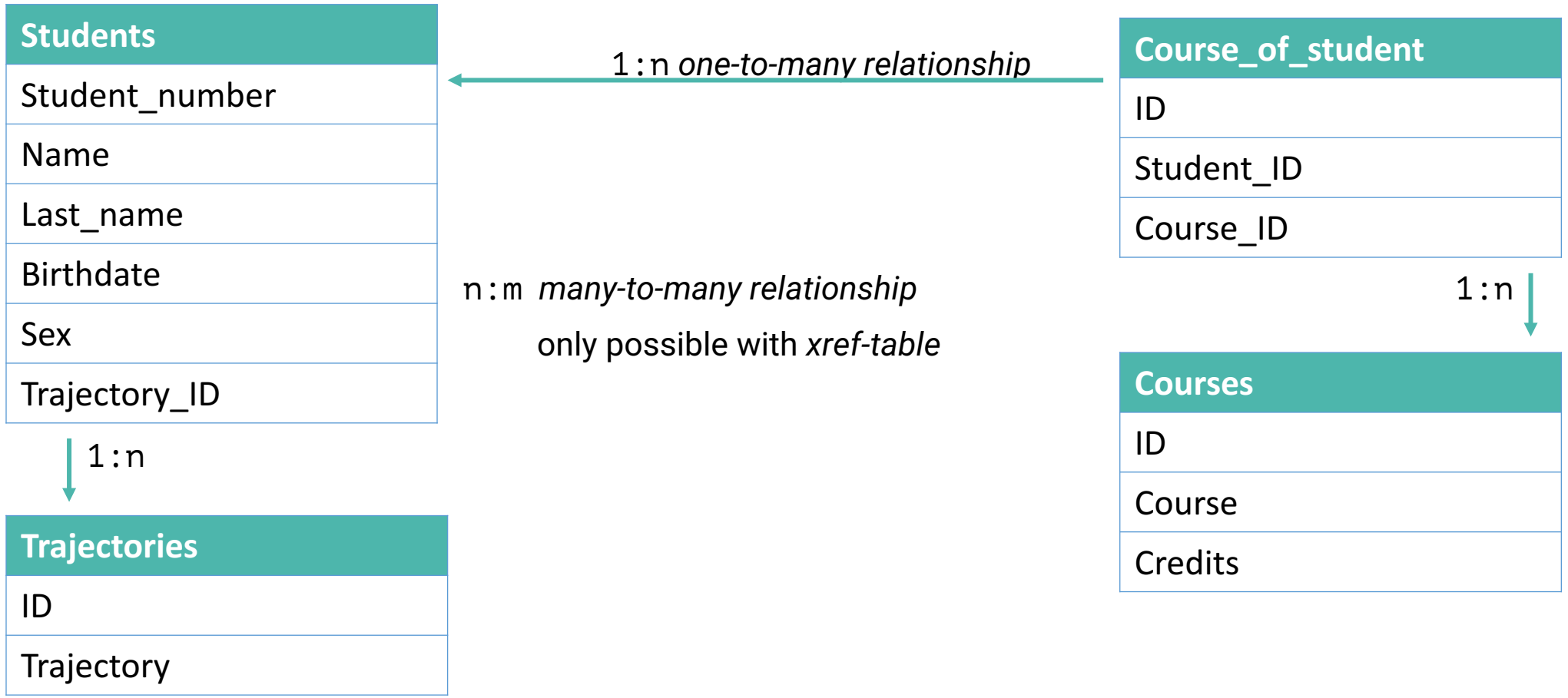
Student_id	Course_ID
0293826	1
0293826	2
0293826	3
0293749	1
0328273	1

Courses

ID	Course
1	Databases
2	Scripting
3	Linux

Relational databases with MySQL

Joins



Database model= Entity Relationship Diagram (ERD)

Relational databases with MySQL

Joins – Information is spread across multiple tables

- ~~Create script that retrieves tables separately~~
- JOIN tables with query

Students
Student_number
Name
Last_name
Birthdate
Sex
Trajectory_ID

Trajectories
ID
Trajectory

Student_number	Name	Last_name	Birthdate	Course
0293826	John	Doe	1991-10-02	Databases
0293749	Mel	Trotter	1991-04-11	Databases
0328273	Bill	Schuette	1990-12-01	Databases
0293826	John	Doe	1991-10-02	Scripting
0293826	John	Doe	1991-10-02	Linux

Data representation does not have to be equal to the way the data is stored

Relational databases with MySQL

Joins

- Retrieve linked rows from different tables with `JOIN`

```
mysql> SELECT * FROM Students
            JOIN Trajectories ON Students.Trajectory_ID = Trajectories.ID;
```

Students

Student_number	Name	Last_name	Birthdate	Trajectory_ID
0293826	John	Doe	1991-10-02	1
0293749	Mel	Trotter	1991-04-11	2
0328273	Bill	Schuette	1990-12-01	2
0324312	Penelope	Tracy	1989-07-24	*NULL*

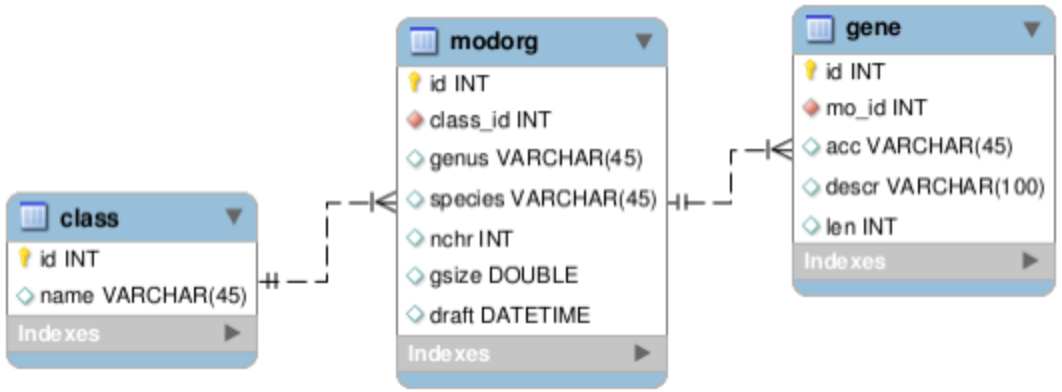
Trajectories

ID	Trajectory
1	FBT
2	MLT
3	TI

Relational databases with MySQL

Joins – back to our database

- `modorg.class_id` is a *foreign key* that references `class.id`
- `gene.mo_id` is a *foreign key* that references `modorg.id`



Relational databases with MySQL

Joins – Cartesian product

- Multiple tables in a query → database server generates all possible combinations
  = Cartesian product

  table `class` has 6 rows
  table `modorg` has 10 rows
  query `SELECT * FROM modorg, class` has 60 rows

Relational databases with MySQL

Joins

- Filtered Cartesian product

```
SELECT * FROM modorg, class WHERE modorg.class_id = class.id;

SELECT * FROM modorg [INNER] JOIN class ON modorg.class_id =
class.id;
```

Relational databases with MySQL

Joins

- Avoid ambiguity

```
mysql> SELECT id, name, genus, species FROM modorg, class WHERE
modorg.class_id = class.id;

ERROR 1052 (23000): Column 'id' in field list is ambiguous
```

- Ambiguous columns must be qualified. And additionally you can choose an alias:

```
mysql> SELECT modorg.id as mo_id, name, genus, species FROM
modorg, class WHERE modorg.class_id = class.id;
```

Relational databases with MySQL

Joins – data source alias

- Join table with itself or select data from subquery
  - Use alias for data source

```
mysql> SELECT a.col, b.col FROM src1 [as] a, src2 [as] b WHERE …
mysql> SELECT a.col, b.col FROM src1 [as] a JOIN src2 [as] b ON
…
```

Relational databases with MySQL

Joins – data source alias

- For each class, give the class name, organism name and date of the organism that was sequenced first

```
mysql> SELECT class_id, min(draft) as dr FROM modorg GROUP BY
class_id;
```

  - add class name: join with table `class`
  - add organism name: join with table `modorg`

Relational databases with MySQL

Joins – data source alias

- Add class name

```
mysql> SELECT name, dr FROM
   (SELECT class_id, min(draft) as dr FROM modorg GROUP BY
   class_id) as s, class WHERE s.class_id=class_id;
```

- add organism name: join with table `modorg`

```
mysql> SELECT name, concat(genus," ",species) as org_name, dr
   FROM (SELECT class_id, min(draft) as dr FROM modorg GROUP BY
   class_id) as s, class, modorg WHERE s.class_id=class.id AND
   s.dr=draft;
```

Relational databases with MySQL

Joins – 4 types

- `INNER JOIN`
  - Only rows present in both tables
- `LEFT JOIN`
  - All rows from left table, even without linked data in right table
- `RIGHT JOIN`
  - All rows from right table, even without linked data in left table
- `OUTER JOIN`
  - All rows from both tables
  - Doesn’t exist in MySQL

Relational databases with MySQL

Joins – `INNER JOIN`

Students

Student_number	Name	Last_name	Birthdate	Trajectory_ID
0293826	John	Doe	1991-10-02	1
0293749	Mel	Trotter	1991-04-11	2
0328273	Bill	Schuette	1990-12-01	2
~~0324312~~	~~Penelope~~	~~Tracy~~	~~1989-07-24~~	*NULL*

Trajectories

ID	Trajectory
1	FBT
2	MLT
~~3~~	~~TI~~

Student_number	Name	Last_name	Birthdate	Trajectory
0293826	John	Doe	1991-10-02	FBT
0293749	Mel	Trotter	1991-04-11	MLT
0328273	Bill	Schuette	1990-12-01	MLT

Relational databases with MySQL

Joins – `LEFT JOIN`

Students

Student_number	Name	Last_name	Birthdate	Trajectory_ID
0293826	John	Doe	1991-10-02	1
0293749	Mel	Trotter	1991-04-11	2
0328273	Bill	Schuette	1990-12-01	2
0324312	Penelope	Tracy	1989-07-24	*NULL*

Trajectories

ID	Trajectory
1	FBT
2	MLT
3	TI

Student_number	Name	Last_name	Birthdate	Trajectory
0293826	John	Doe	1991-10-02	FBT
0293749	Mel	Trotter	1991-04-11	MLT
0328273	Bill	Schuette	1990-12-01	MLT
0324312	Penelope	Tracy	1989-07-24	*NULL*

Relational databases with MySQL

Joins – `RIGHT JOIN`

Students

Student_number	Name	Last_name	Birthdate	Trajectory_ID
0293826	John	Doe	1991-10-02	1
0293749	Mel	Trotter	1991-04-11	2
0328273	Bill	Schuette	1990-12-01	2
~~0324312~~	~~Penelope~~	~~Tracy~~	~~1989-07-24~~	*NULL*

Trajectories

ID	Trajectory
1	FBT
2	MLT
3	TI

Student_number	Name	Last_name	Birthdate	Trajectory
0293826	John	Doe	1991-10-02	FBT
0293749	Mel	Trotter	1991-04-11	MLT
0328273	Bill	Schuette	1990-12-01	MLT
NULL	*NULL*	*NULL*	*NULL*	TI

Relational databases with MySQL

Joins – `OUTER JOIN`

Students

Student_number	Name	Last_name	Birthdate	Trajectory_ID
0293826	John	Doe	1991-10-02	1
0293749	Mel	Trotter	1991-04-11	2
0328273	Bill	Schuette	1990-12-01	2
0324312	Penelope	Tracy	1989-07-24	*NULL*

Trajectories

ID	Trajectory
1	FBT
2	MLT
3	TI

Student_number	Name	Last_name	Birthdate	Trajectory
0293826	John	Doe	1991-10-02	FBT
0293749	Mel	Trotter	1991-04-11	MLT
0328273	Bill	Schuette	1990-12-01	MLT
0324312	Penelope	Tracy	1989-07-24	*NULL*
NULL	*NULL*	*NULL*	*NULL*	TI

Relational databases with MySQL

Joins – characteristics

- `*` is used to select all columns from all tables
- Use *`tbl.col`* to specify column
- Very inefficient
  - A lot of memory, a lot of time
- Use `ON` to specify linked columns

Relational databases with MySQL

Execution order

1. Input columns are determined
    a. JOIN clause
2. `WHERE` – input columns are filtered
3. `GROUP BY` – sorting & grouping of filtered input
4. Aggregation functions are calculated
5. `HAVING` – aggregation functions are filtered
6. `ORDER BY` – output is sorted
7. `LIMIT/OFFSET` – output is chopped

Relational databases with MySQL

Exercises

- Experiment with the different types of JOINs
- For all rows in table gene, show
  - Organism name
  - Class name
  - Accession
  - Length
  - Description of the gene

Relational databases with MySQL

Efficiency and speed

- Complex queries tend to become
  - Large system load
  - Slow interface, user has to wait
- Mind your choices of column types and included columns in the query
- Frequently used queries can be sped up
  - Views
  - Indices
  - Allow redundancy

Relational databases with MySQL

Views

- Re-use same query
- Query can be saved as a special table-like object
- Usually read-only data source
- Speed gain depends on use case
- MySQL
  - Virtual table
  - Used to serve up data in an orderly fashion
- Oracle i.a.
  - Materialized view: result of a SELECT query is stored
    - Very efficient

Relational databases with MySQL

Views

- Create a new view

```
mysql> CREATE VIEW viewname as SELECT …
```

- Use a view as a table

```
mysql> SELECT … FROM viewname WHERE … ORDER BY …
```

Relational databases with MySQL

Exercises

- Create a view (`genevw`) from the query in the previous exercise
- Select all genes containing hemoglobin in the description and sort the result set by gene length.
  Next:
  - What is the minimum and maximum gene length?
  - What is the average gene length?
  - And the standard deviation?
- Does the view show up when using

```
mysql> show tables;
```

Relational databases with MySQL

Index

- Quickly find certain rows
- Stored separately
- Golden rule: Use indices on columns you will use in your `WHERE` clause
- Only 1 index per query is used

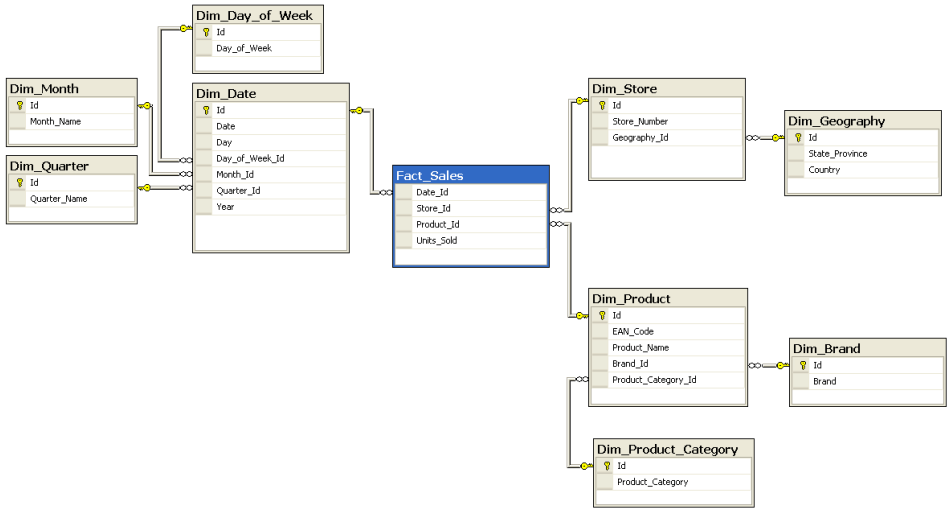
```
CREATE [UNIQUE|FULLTEXT|SPATIAL] INDEX index_name ON tbl (col);
```

- Foreign key
  - Index used to speed up `JOIN` queries
  - Not essential for `JOIN` queries

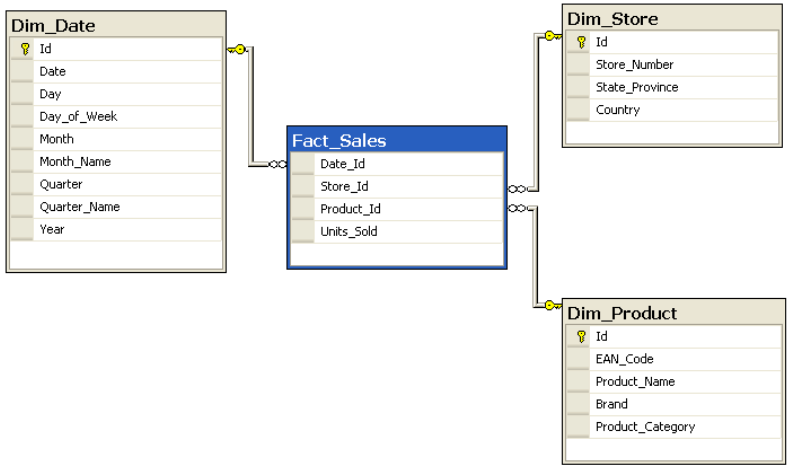
Relational databases with MySQL

Allow redundancy

- Schema with least redundancy isn't always the quickest
- Allow redundancy to reduce number of `JOIN`s
- E.g. Data warehouse with real-time reporting
  - High efficiency is required
- Snowflake vs Star schema



Snowflake



Star

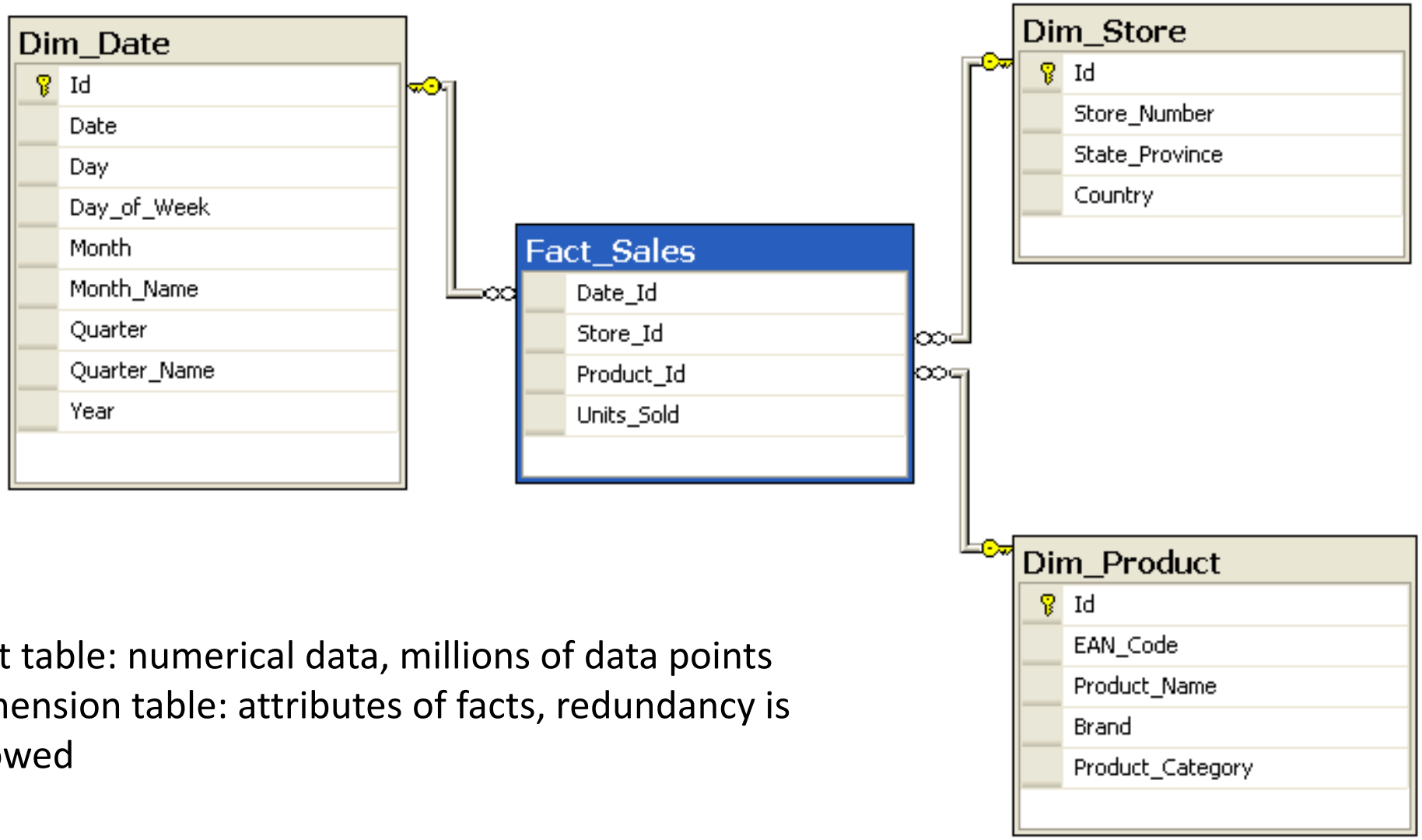
Relational databases with MySQL

SNOWFLAKE	STAR
No redundancy	Redundant data
Easy to maintain and change	Less easy to maintain/change
Complex queries	Lower query complexity
Slower (more JOINs)	Faster
Uses less space	Uses more space (data is stored twice or more)
Bottom up	Top down

```
SELECT
        B.Brand,
        G.Country,
        SUM(F.Units_Sold)
FROM Fact_Sales F
INNER JOIN Dim_Date D ON F.Date_Id = D.Id
INNER JOIN Dim_Store S ON F.Store_Id = S.Id
INNER JOIN Dim_Geography G ON S.Geography_Id = G.Id
INNER JOIN Dim_Product P ON F.Product_Id = P.Id
INNER JOIN Dim_Brand B ON P.Brand_Id = B.Id
INNER JOIN Dim_Product_Category C ON
P.Product_Category_Id = C.Id
WHERE D.Year = 1997 AND C.Product_Category = 'tv'
GROUP BY B.Brand, G.Country
```

```
SELECT   P.Brand,
         S.Country AS Countries,
         SUM(F.Units_Sold)
FROM Fact_Sales F
INNER JOIN Dim_Date D ON (F.Date_Id = D.Id)
INNER JOIN Dim_Store S ON (F.Store_Id = S.Id)
INNER JOIN Dim_Product P ON (F.Product_Id = P.Id)
WHERE D.Year = 1997 AND P.Product_Category = 'tv'
GROUP BY P.Brand, S.Country
```

Relational databases with MySQL



- Fact table: numerical data, millions of data points
- Dimension table: attributes of facts, redundancy is allowed

Relational databases with MySQL

Database backup

- Dump a complete database into a tekst file
  `$ mysqldump [opt] db > db.sql`
- Includes
  - Statements for creating the database if the option `--databases` is used
  - Statements for creating tables, views, …
  - Statements for inserting data
- Restore the database (may need to create first)
  `$ mysql db < db.sql`

Relational databases with MySQL

Exercises

- Create a separate dump file for each of your own databases
- Check the contents of each file
- For the bold ones
    - Drop your databases and recreate them using your dump files