BIT05 - Databanktechnologie

Jasper Anckaert

Lecture 2 – Relational databases 2

Previously

- Database
  - Collection of data that needs to be stored
  - Structured
  - Used everywhere
- Database system
  - Hardware – data – software – users
  - Storage space – quick – little redundancy – secure – clear structure
- Database Management System (DBMS)
  - Software application that interacts with the user, other applications, and the database itself to capture and analyse data
  - Storage – Retrieval – Manipulation – Authentication & authorization
- Relational databases – Relational Database Management System (RDBMS)
  - Enforce data integrity
  - Enforce referential intigrety
  - Rules of E. Codd

Previously

- MySQL
  - Install, connect to and secure server
  - User – host – database – table
  - Privileges
  - Options file

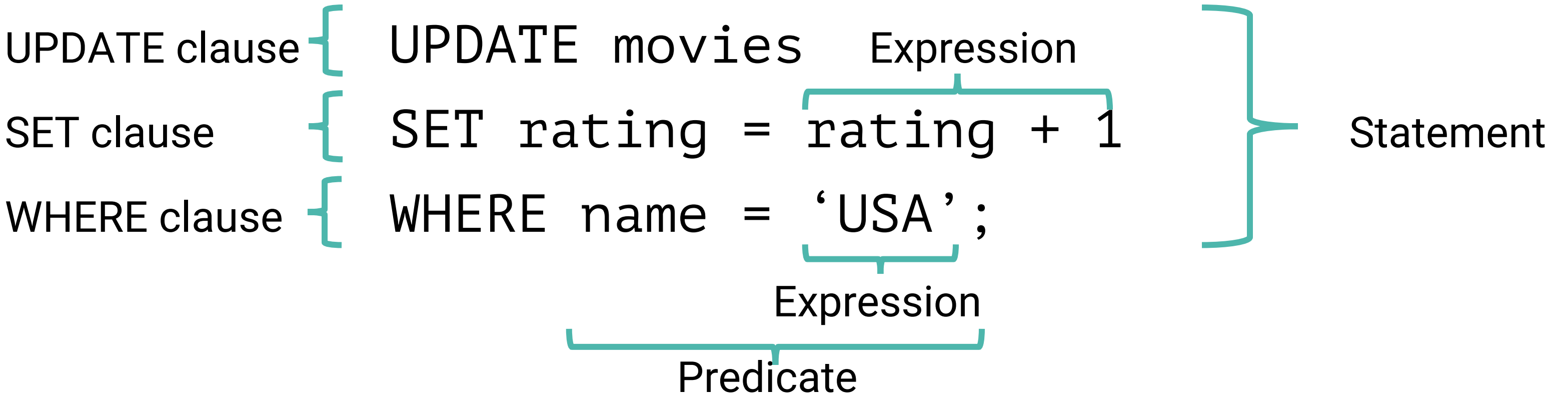
  - Create database
  - Grant privileges
  - Show databases, tables columns, create statement

Previously

- `SQL`
  - Data definition language
    - Statements to design database
    - `CREATE`, `ALTER`, `DROP`, …
  - Data manipulation language
    - Statements to manage data
    - CRUD
    - `SELECT`, `INSERT`, `UPDATE`, `DELETE`
  - Data control language
    - Statements to manage database rights
    - `GRANT`, `REVOKE`

Previously

SQL: Structured Query Language



```
UPDATE movies
SET rating = rating + 1
WHERE name = ‘USA’;
```

UPDATE clause

SET clause

WHERE clause

Expression

Expression

Predicate

Statement

Previously

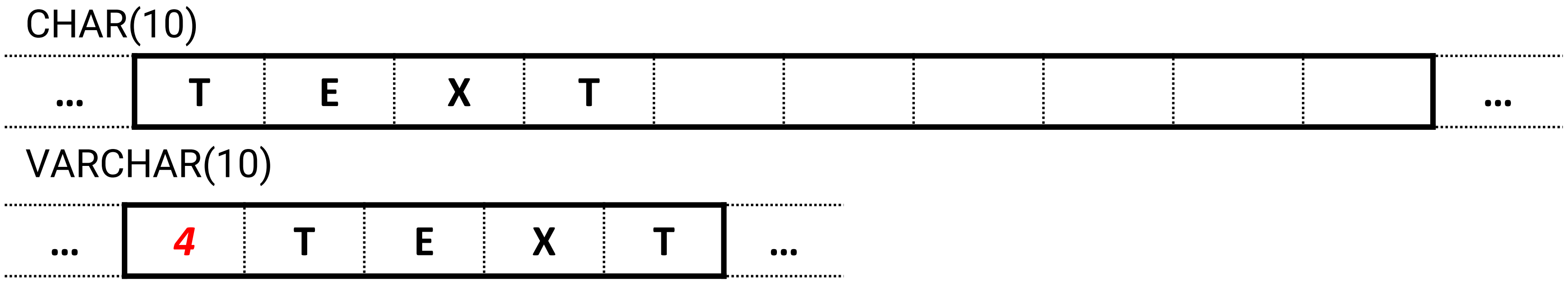
Column types

- 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

Previously

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



CHAR(10)

… T E X T …

VARCHAR(10)

… 4 T E X T …

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

Previously

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

Previously

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

Previously

- INSERT

```
INSERT INTO tbl (col1, col2) VALUES (val1, val2);
```

- SELECT

```
SELECT columns FROM tbl;
```

- ORDER BY

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

- Calculated rows
  - Built in functions for numbers, strings, dates
- Column aliases
  - Can be used in the `ORDER BY` clause
- WHERE

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

- NULL values

```
SELECT … WHERE col IS [NOT] NULL;
SELECT ifnull(col, value) …
```

Previously

- AND, OR, NOT, XOR
  - Boolean logic
- DISTINCT

  `SELECT DISTINCT(`*`cols`*`) FROM …`
- LIMIT, OFFSET

  `SELECT … LIMIT `*`n`*` [OFFSET `*`r`*`];`
- Aggregation
  - Built in functions e.g. `count()`, `sum()`, `min()`, `max()`, …
- GROUP BY

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

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

Previously

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

- Return a list of the 3 biotypes with the most genes. Which biotype is a close fourth?
- Return a list of the number of genes per status
- Combining the 2 previous results, which biotype is most known?
- Select only those biotypes that cover at least 3% of the human genome (hint: ‘size’ in previous exercise, human genome is approx. 3 billion bp) and return this percentage.

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

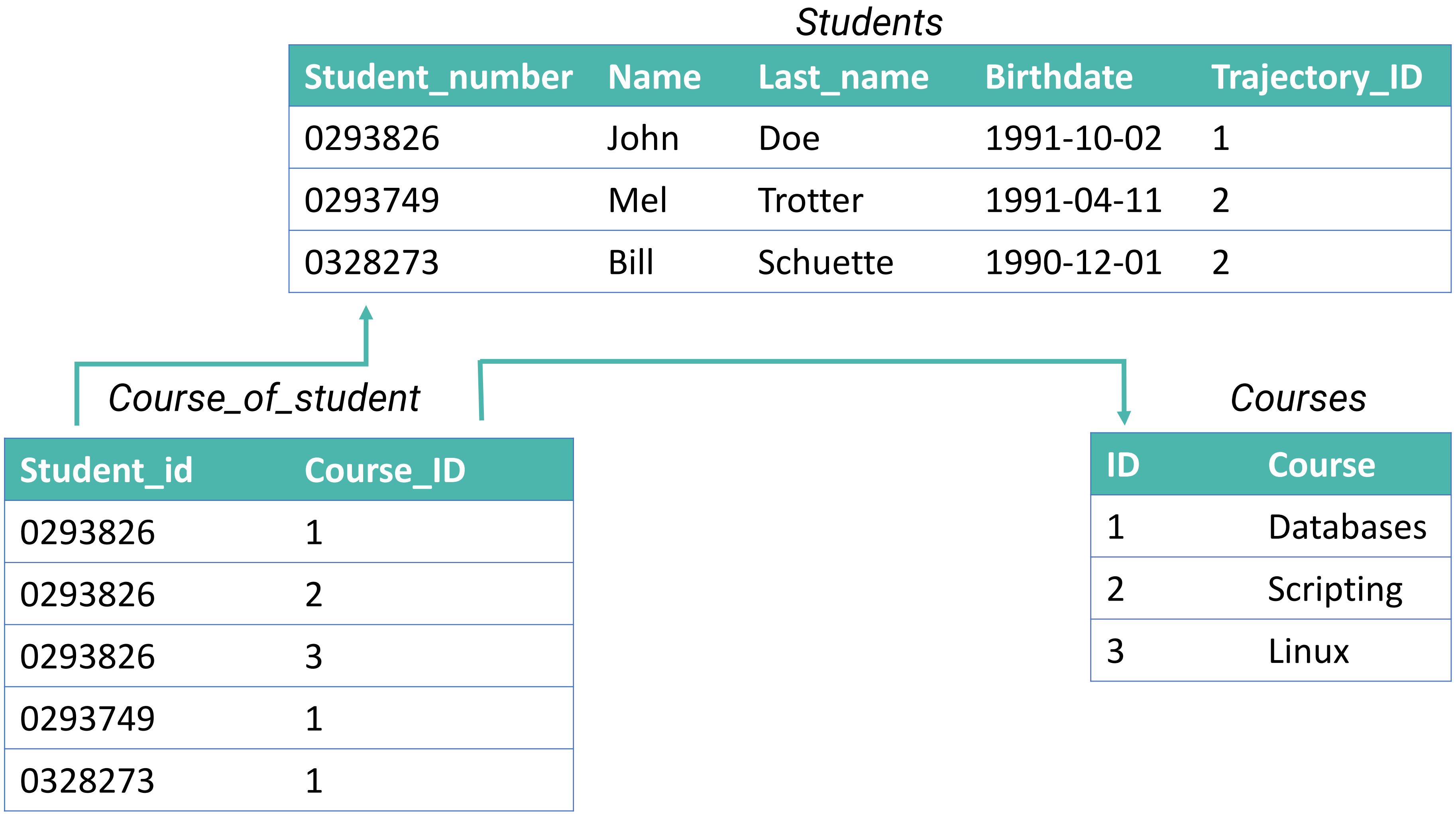
- Important: Data types of linked columns have to be equal! Foreign key is typically primary key of other table

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

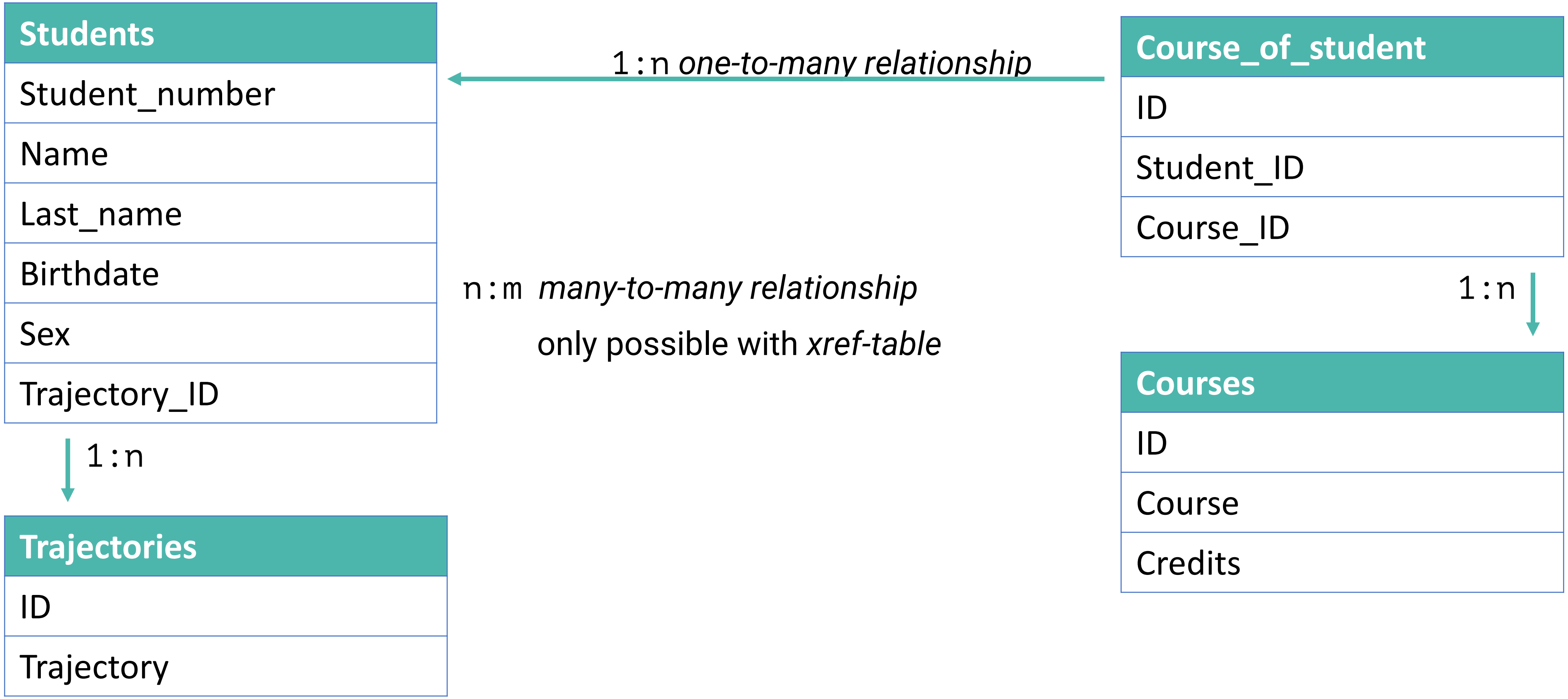
Student_id	Course_ID
0293826	1
0293826	2
0293826	3
0293749	1
0328273	1

Courses

ID	Course
1	Databases
2	Scripting
3	Linux

Relational databases with MySQL

Joins



Students
Student_number
Name
Last_name
Birthdate
Sex
Trajectory_ID

Course_of_student
ID
Student_ID
Course_ID

Trajectories
ID
Trajectory

Courses
ID
Course
Credits

`1:n` *one-to-many relationship*

`n:m` *many-to-many relationship*

only possible with *xref-table*

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*

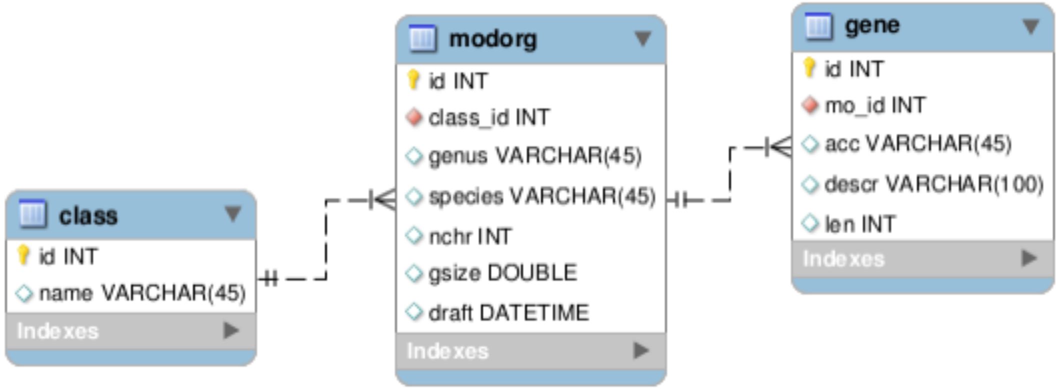
Trajectories

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`



class
id INT
name VARCHAR(45)
Indexes

modorg
id INT
class_id INT
genus VARCHAR(45)
species VARCHAR(45)
nchr INT
gsize DOUBLE
draft DATETIME
Indexes

gene
id INT
mo_id INT
acc VARCHAR(45)
descr VARCHAR(100)
len INT
Indexes

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 …
```

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

Exercises

- For all rows in table `gene`, show
  - Organism name
  - Class name
  - Accession
  - Length
  - Description of the gene

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

Excercises

- Switch back to the `bioinf_testdb` database
  - Experiment with the 3 types of `JOIN` on your own tables
  - Examine the `gene`, `transcript` and `exon` table, how are they connected?
    - How many transcript does the *MALAT1* gene have?
    - Return the position of the exons of transcript *237999*
  - Return the transcripts of the *TP53* gene
  - Return their exons as well
  - Find the longest spliced transcript of *TP53* (taking into account the intron-exon structure)
  - How many exons does each transcript have?

Relational databases with MySQL

Excercises

- What is the name of the gene associated with transcript 260392?
- A mutation was found on chromosome 20, position 44002590. Which gene(s) overlap(s) with this position?
    - Select only those genes that have exons that overlap with this mutation. Which genes are they?
- A biotype column can be found in both the gene and transcript table. Are there transcripts that have a different biotype from the gene they're part of? What are their names?
    - Does the same go for status?
- Which chromosome has the most genes and how many are there?
- Which exon is the largest in the genome and how many base pairs are there?
- Which transcript has the most exons and how many are there?

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

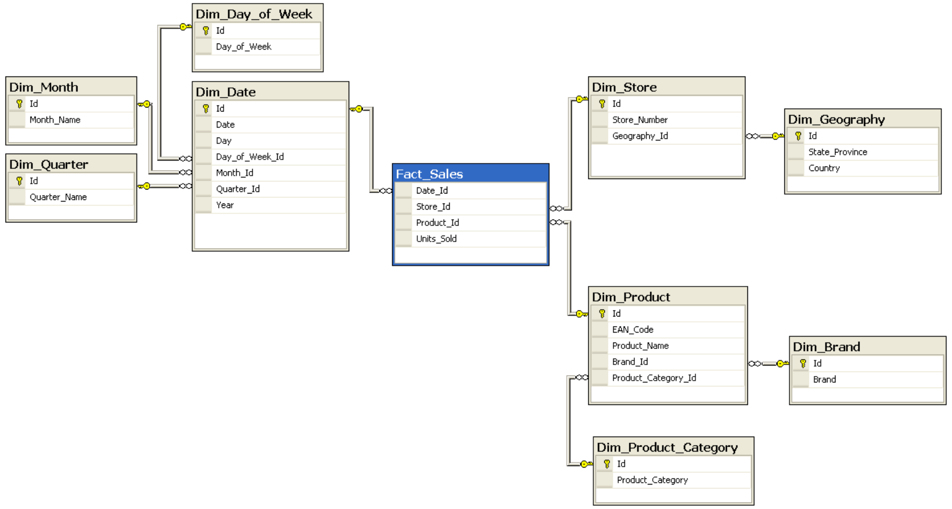
  ```
  CREATE [UNIQUE|FULLTEXT|SPATIAL] INDEX index_name ON tbl (col);
  ```

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

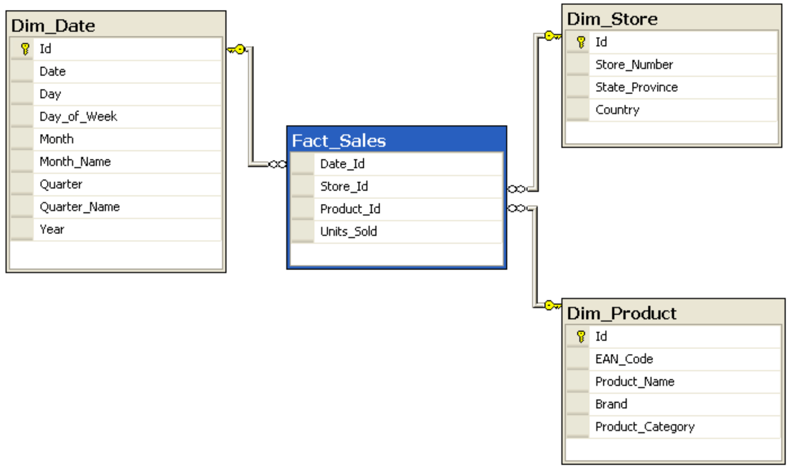
Relational databases with MySQL

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

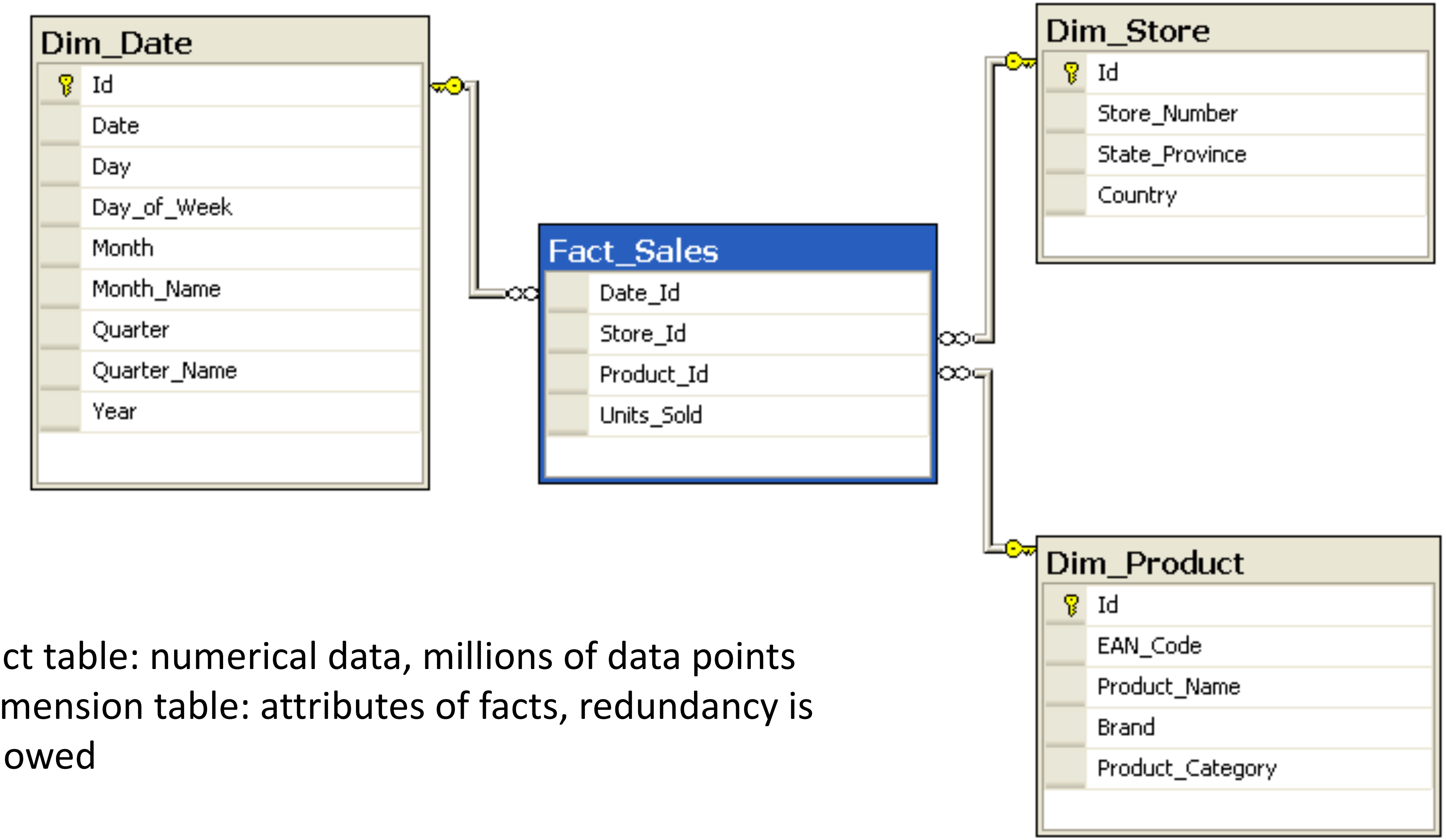
Relational databases with MySQL

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



Dim_Date
- Id
- Date
- Day
- Day_of_Week
- Month
- Month_Name
- Quarter
- Quarter_Name
- Year

Fact_Sales
- Date_Id
- Store_Id
- Product_Id
- Units_Sold

Dim_Store
- Id
- Store_Number
- State_Province
- Country

Dim_Product
- Id
- EAN_Code
- Product_Name
- Brand
- Product_Category

- 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

Relational databases with MySQL

Rehearsal exercises

- Create a new database for your lab and include following data
  - All trainings
    - Subject, duration
  - All lab members
    - Name, lastname, birth_date, training
  - All equipment
    - Name, manufacturer, purchase_date
  - All experiments
    - Name, performed_by, equipment_used, date
  - All results
    - Directory, experiment, status
- Fill with some data

```
CREATE TABLE Kit_order (
        order_number CHAR(16) NOT NULL,
        manufacturer VARCHAR(255),
        kit_name VARCHAR(255),
        supplier VARCHAR(255),
        PRIMARY KEY (order_number),
        FOREIGN KEY (manufacturer, kit_name)
        REFERENCES Kit(manufacturer, name),
        FOREIGN KEY (supplier)
        REFERENCES Supplier(supplier_name));
```

```
CREATE TABLE Kit (
        manufacturer VARCHAR(255) NOT NULL,
        name VARCHAR(255) NOT NULL,
        kit_cost DECIMAL(6,2),
        buffer VARCHAR(255),
        buffer_conc FLOAT,
        enzyme VARCHAR(255),
        enzyme_conc FLOAT,
        nucl_mix VARCHAR(255),
        nucl_conc FLOAT,
PRIMARY KEY (manufacturer, name));
```

Relational databases with MySQL

Rehearsal exercises (part 2)

- In your newly created database, search for
  - All experiment equipment purchased after 1st of January 1985
  - The number of experiments each lab member conducted
  - A list with all equipment used in a successful experiment
  - A list with all lab members that failed an experiment
  - Who followed wich trainings?
    - Number of participants per training