An introduction to relational database management system

Fabrice Rossi

CEREMADE
Université Paris Dauphine

2021

Database Management Systems

Database (DB)

An organized collection of data

Database Management System (DBMS)

A software used to manage databases, namely

- creating a database
- maintaining a database
- interacting with a database

DBMS versus Files

File-Based data management

- data in CSV files (or other format)
- management with e.g. Python + Pandas (or R + tidyverse)

DBMS versus Files

File-Based data management

- data in CSV files (or other format)
- management with e.g. Python + Pandas (or R + tidyverse)

Advantages of DBMS

- meta data management
- integrity constraints
- security (access control)
- concurrency control
- etc.

DBMS components

DBMS architecture

- storage manager
  - handles saving data and meta data on the storage system
  - manages part of the concurrency aspects
- query processor
  - core part of the DBMS
  - handles computation on the database (e.g. search, insertion, deletion, modification, aggregation)
  - optimize queries for efficiency
- high level tools
  - meta data management
  - connection and security manager

DBMS components

DBMS languages

- data definition language (DDL)
  - used to specify a data model
    - data types
    - links between data
  - meta data management
- data manipulation language (DML)
  - used to query a database
  - specifies computation on a database

Types of DBMS

Data model categorization

- old approaches: hierarchical DBMS and network DBMS
- current standard: relational DBMS
  - based on a relational data model
  - typical based on SQL (Structured Query Language, sequel)
- niche approach: object-oriented DBMS
- large scale or unstructured data: NoSQL
  - not-only SQL (or more specifically not relational)
  - document stores
  - graph oriented databases
  - column oriented databases

In this course

An introduction to Relational DBMS

In this course

An introduction to Relational DBMS

What's missing?

- A lot!
- DBMS are a very rich and complex subject
  - numerous theoretical aspects
  - getting under the hood is yet another subject
  - practical aspects are also very complex

Outline

Outline

Data Management

Specifying the data

- ideally the first step of a data oriented project
- important questions
  - what are the objects under study?
  - how are they described?
  - how are they related?

Two steps process

1. conceptual/formal model
2. implementation in a meta data oriented language

Entity Relationship Model

Proposed by Peter Chen in 1976

Concepts

- Entity: uniquely identified object under study (e.g. a person)
- Relationship: a way to relate entities (e.g. a has access to b)
- Attribute: a property of an entity or of a relationship. An attribute has a domain (the set of values it can take)
- an ER model describes types, e.g. entity type, not values

Example

Loan application data set

- https://relational.fit.cvut.cz/dataset/Financial
- 8 tables including
  - client table
  - account table
  - credit card table
  - loan table
  - etc.

Example

(part of the) Client table

client_id	gender	birth_date
1	F	1970-12-13
2	M	1945-02-04
3	F	1940-10-09
4	M	1956-12-01
5	F	1960-07-03

ER model

- entity type: client
- attributes
  - gender with domain F and M
  - birth_date with a date domain
  - client_id

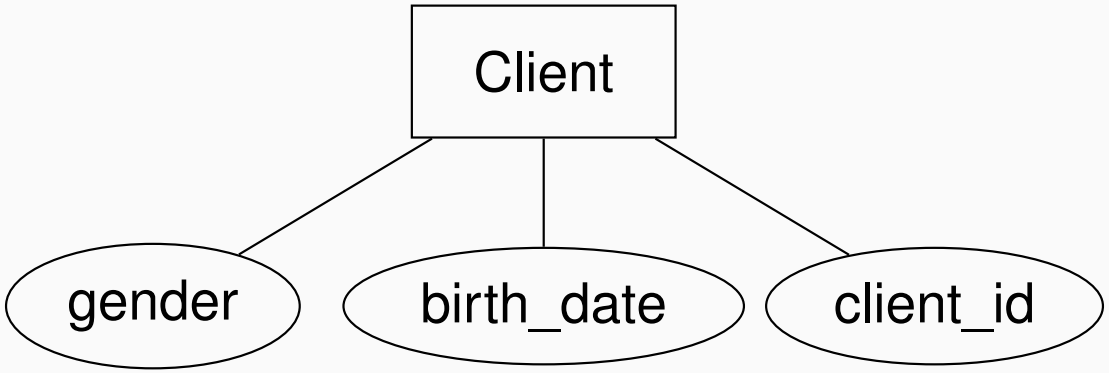
Example

(part of the) Client table

client_id	gender	birth_date
1	F	1970-12-13
2	M	1945-02-04
3	F	1940-10-09
4	M	1956-12-01
5	F	1960-07-03

ER model

- entity type: client
- attributes
  - gender with domain F and M
  - birth_date with a date domain
  - client_id



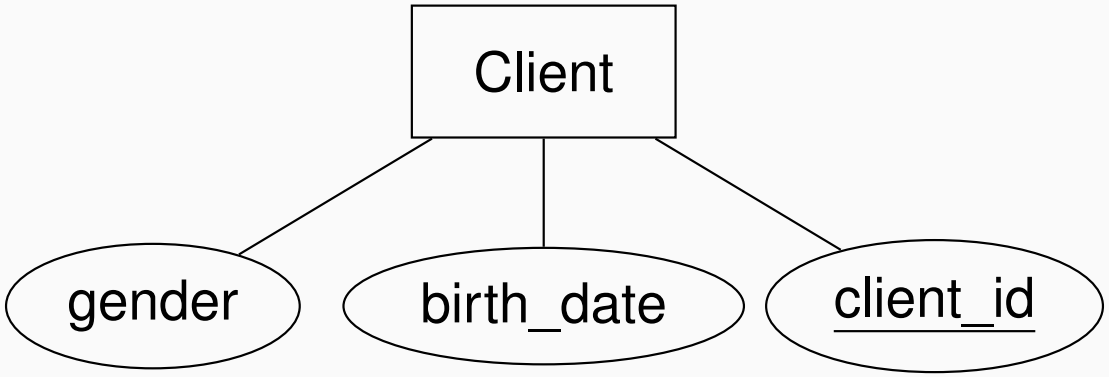
Example

(part of the) Client table

client_id	gender	birth_date
1	F	1970-12-13
2	M	1945-02-04
3	F	1940-10-09
4	M	1956-12-01
5	F	1960-07-03

ER model

- entity type: client
- attributes
  - gender with domain F and M
  - birth_date with a date domain
  - client_id key attribute



Graphical representation

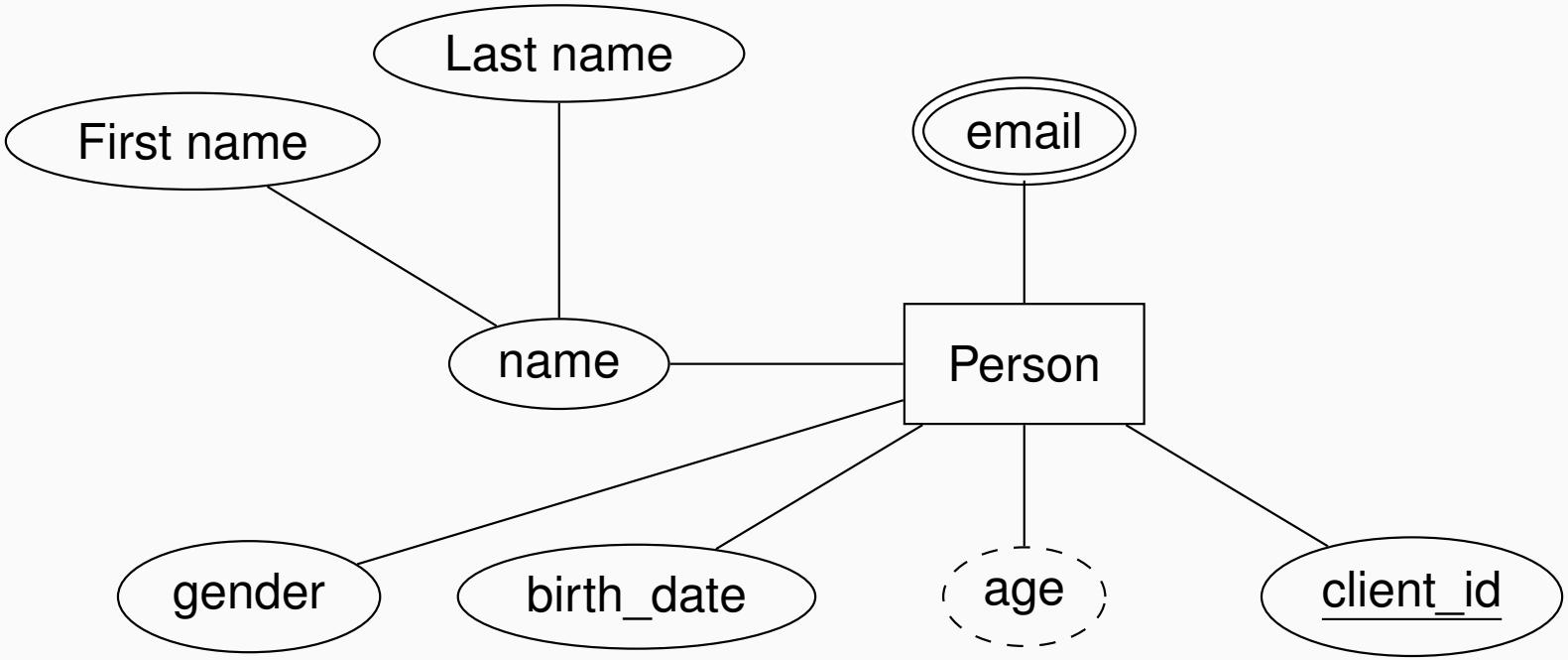
Entity type

Entity types are represented by rectangles link to attributes

Attribute type

- an ellipse per type
- key attribute type
  - a unique identifier of the corresponding entity
  - underlined in the representation
- composite attribute type
  - can be decomposed into sub-attributes
  - linked ellipses
- derived attribute type
  - can be computed from another one (e.g. age from birth date)
  - dashed border
- multi-valued attribute type
  - several values are authorized
  - double border

Example


Last name
First name
email
name
Person
gender
birth_date
age
client_id

Typical domains

Numerical

- integers
- decimal numbers
- possible constraints: positive numbers, number of significant digits, etc.

Temporal

- dates
- times

Textual

- words
- codes (such as post codes)
- structured strings (e.g. emails)

Others

- truth values (boolean)
- binary content (such as images)

Relationship

Principle

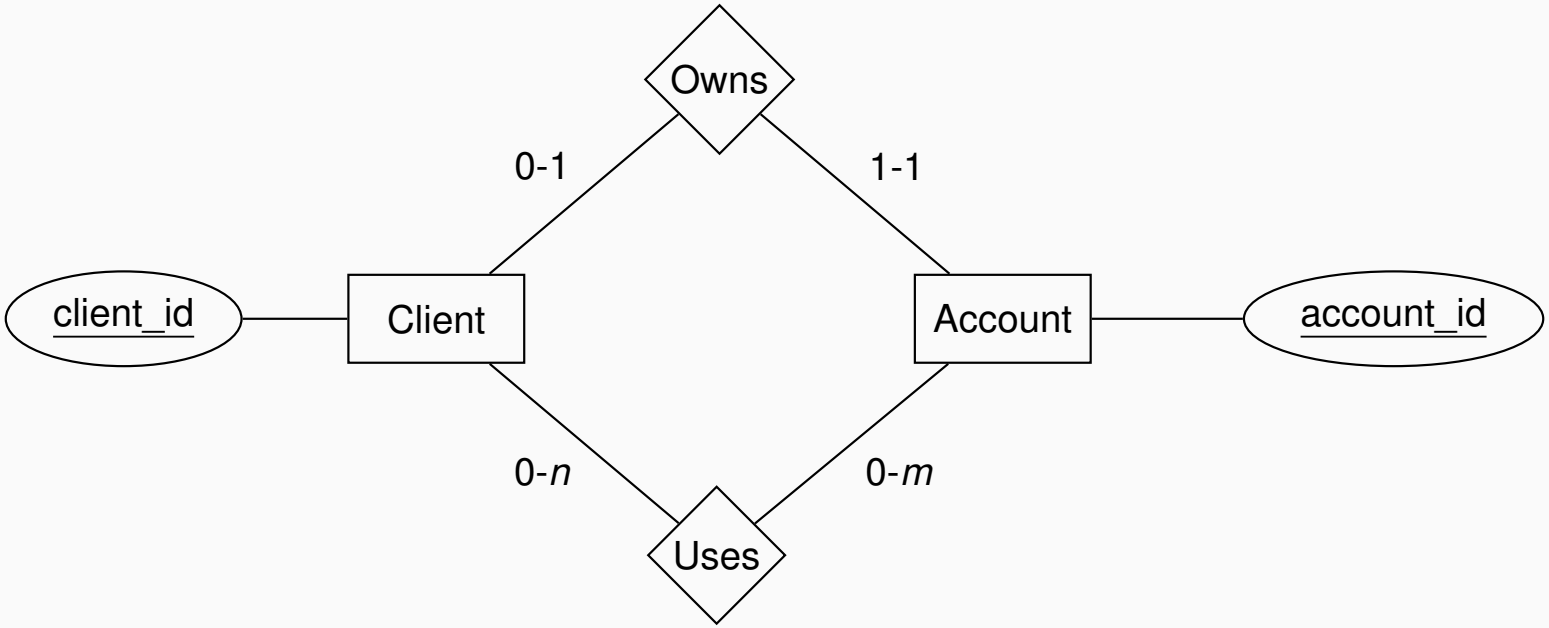
- a relationship represents an association between at least two entities
- it can have attributes
- it is characterized by cardinalities
  - minimum and maximum number of relationships to which a given entity can participate
  - asymmetric
- a relationship type is graphically represented by a rhombus

Example

Loan application data set

- client entities and account entities
- relationships
  - a client can be the owner of an account
  - a client can be allowed to use an account
- cardinalities
  - owner:
    - each account has exactly one owner
    - each client can own at most one account (in this database)
  - user:
    - each account may have some users
    - each client can be the user of some accounts

Example



Outline

Relational Model

History

- invented by Edgar F. Codd in 1969-1970
- based on a mathematical model, the relational algebra
- associated relational calculus

Main concepts

- a *domain*: a set
- a *relation*: a (finite) subset of $\prod_{i=1}^{n}(D_i \cup \{NULL\})$ where each D_i is a *domain* and *NULL* a special value (for missing values)
- a *tuple*: an element of a *relation*
- a *database*: a collection of *relations*

Examples

Clients

- Domains:
  - $\mathbb{N}^+$: positive integers
  - $G = \{F, M\}$: genders
  - D: dates
- A *client* relation:
  - a subset of $\mathbb{N}^+ \times G \times D$
  - a set of tuples such as $(1, F, 1970{-}12{-}13)$

Accounts

- Domains:
  - $\mathbb{N}^+$: positive integers
  - D: dates
- An *account* relation:
  - a subset of $\mathbb{N}^+ \times \mathbb{N}^+ \times D$
  - a set of tuples such as $(1, 8, 1995{-}03{-}24)$

Relation type/variable

Specifying relations

- somewhat imprecise vocabulary
- a relation (value) $r \in \prod_{i=1}^{n}(D_i \cup \{NULL\})$
- a relation type/variable (relvar)
  - a *named* specification of the relation
  - domains and names: *column types*

Notation

- a relation type/variable is denoted $R(A_1, \ldots, A_n)$:
  - R: name of the relation type
  - A_k: column type names
- e.g. client(client_id, gender, birth_date)
- and account(account_id, district_id, date)

ER and relational model

Correspondence

- Entity type: relation type/variable, relvar
- Entity: tuple
- Attribute type: column type (frequently called attribute type)
- Attribute: value in a tuple

More about ER $\rightarrow$ relational later

Keys

Superkeys and keys

- a relation is a *set* $\Rightarrow$ each tuple is *unique*
- a *superkey*
  - any subset of the attribute types such that no tuples of the relation share exactly the same values for these attribute types
  - default superkey: all the attribute types!
- a *key*: a minimal superkey (removing attribute types removes it superkey status)

France data set

- departement(region_id,departement_id,name)
- superkeys
  - (region_id,departement_id,name)
  - (departement_id,name)
  - (departement_id): a key!
  - (name): another key!

Primary and foreign keys

Primary key

- several keys: *candidate* keys
- one of the is *the primary key*
- other are *alternative* keys

Foreign keys

- a set of attribute types FK in a relation type R_1 is a *foreign key* of R_1 if
  1. a candidate key in a relation R_2 has exactly the same domains as the ones of the attribute types in FK
  2. values on FK in a tuple in r_1 are either *NULL* or occur in a tuple in r_2
- in simple terms: a foreign key links R_1 to R_2

Example

Relvar

- client(**client_id**, gender, birth_date)
- account(**account_id**, district_id, date)
- disp(**disp_id**, client_id, account_id, type) with type$\in \{\mathit{OWNER}, \mathit{DISPONENT}\}$

Many keys

- bold and underlined: primary keys
- underlined: foreign keys
- district_id is also a foreign key but for another relation
- in this database, the relation between a client and an account is unique and thus (client_id, account_id) is candidate key

Integrity Constraints

Relational constraints

When implemented the relational model must enforce some constraints

- Domain integrity: the values in a tuple are unique values from the associated domain or *NULL* (if allowed, i.e. if the attribute type is *nullable*)
- Key constraint: every relation has a primary key
- Entity integrity: values in the primary key cannot be *NULL*
- Referential integrity: foreign keys can be either *NULL* or must refer to an existing tuple

ER to relational

Mapping

- ER models are abstract
- must be mapped to a concrete database model
- the relational model is close enough to ER to enable a simple mapping strategy

Principles

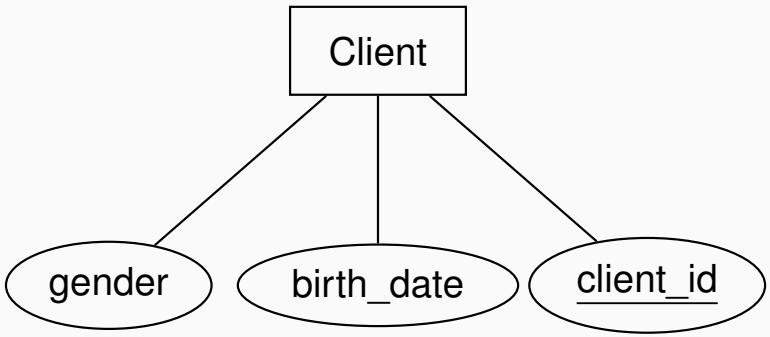
- Entity type $\rightarrow$ relation (type)
- Simple attribute type $\rightarrow$ column type
- Key attribute type $\rightarrow$ primary or alternative key
- All the rest (relationship types and complex attribute types) $\rightarrow$ relation (type) and keys

Mapping Entity Types

With simple attribute types

- direct mapping
- an entity type is mapped to a relation type
- each attribute type corresponds to a column type
- a key is mapped to a primary or alternative key
- composite attribute types are mapped to a set of column types

ER model



Relation type

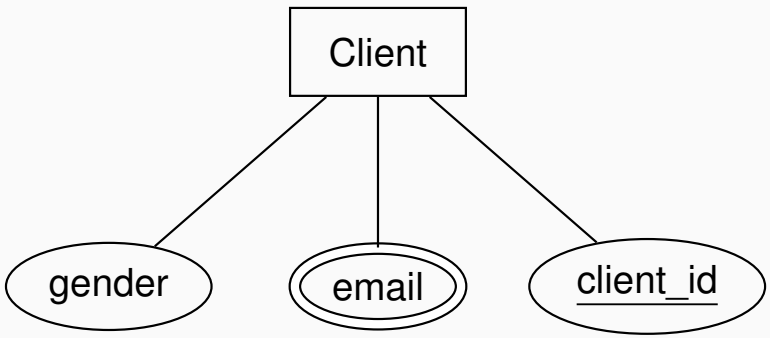
Client(gender, birth_date, **client_id**)

Multi-valued attribute types

Method

- a multi-valued attribute type cannot be mapped to a column type because of the domain integrity constraint
- representation via a relation type
  - a relation type per multi-valued attribute type
  - a column type for the attribute
  - a foreign key to map back the attribute to the entity

ER model



Relation types

- Client(gender, **client_id**)
- ClientEmail(**email**, client_id)

Mapping relationship types

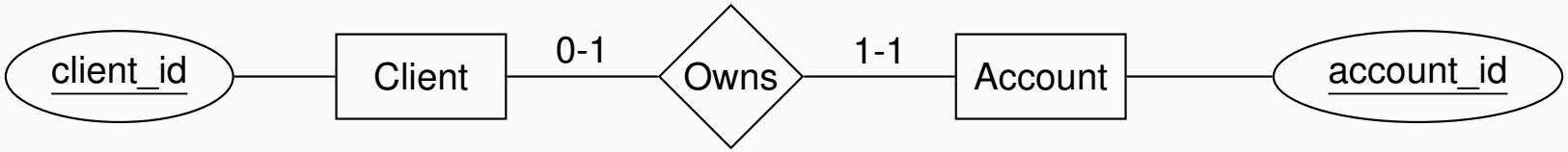
Principles

- the mapping depends on the cardinalities of the relationship type
- 0-1 cardinalities on a least one side foreign key
- other cases: foreign keys or specific relational type

One to one (1:1) relationship type

- each side is 0-1 or 1-1
- mapped to a foreign key:
  - in the 1-1 relation type if it exists with a non *nullable* column type
  - *nullable* if both side are 0-1

Example



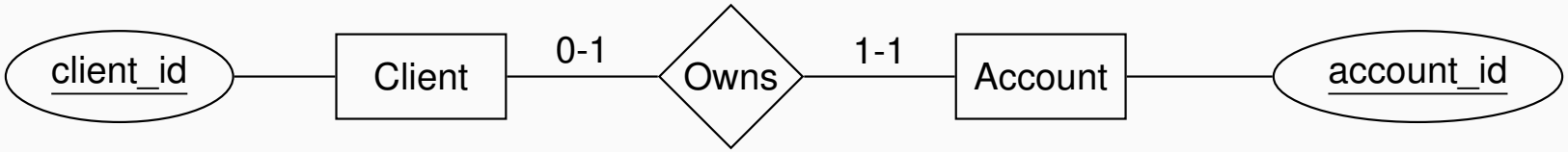
Relation types

- Client(**client_id**)
- Account(**account_id**,client_id)
- client_id is non *nullable* in Account

Inferior alternative

- Client(**client_id**,account_id)
- account_id is *nullable* in Client
- Account(**account_id**)

Example



Relation types

- Client(**client_id**)
- Account(**account_id**,client_id)
- client_id is non *nullable* in Account

Cardinalities

- Owns $\leftrightarrow$ Account is enforced
- but an account can be shared by several clients
- Client $\leftrightarrow$ Owns is $0 - m$

Inferior alternative

- Client(**client_id**,account_id)
- account_id is *nullable* in Client
- Account(**account_id**)

Cardinalities

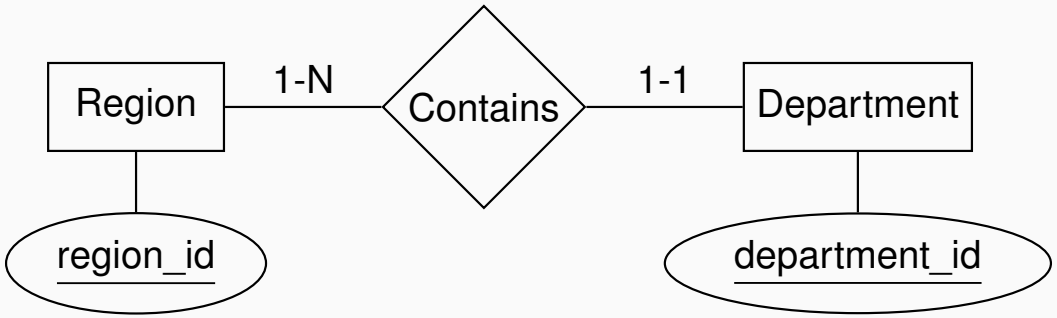
- Client $\leftrightarrow$ Owns is enforced
- but an account can be assigned to any number of clients
- Owns $\leftrightarrow$ Account is $0 - m$

Higher cardinalities

One to *N* (1:*N*) relationship type

- mapped to a foreign key in the 1 side entity relation type
- *nullable* if this side is $0-1$, non nullable if it is $1-1$
- the minimal cardinality on the *N* side cannot be enforced

Example



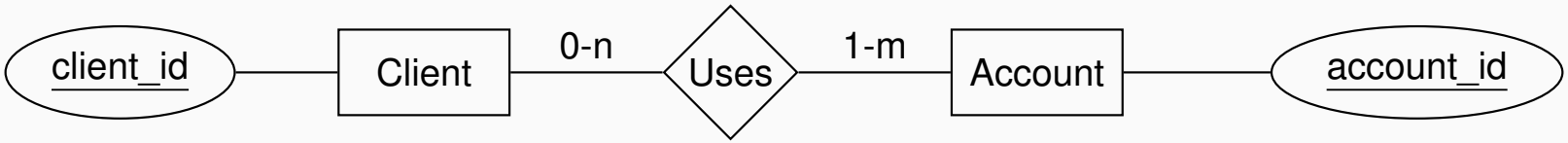
- Department(**departement_id**,region_id)
- Region(**region_id**)

Higher cardinalities

M to *N* (*M*:*N*) relationship type

- mapped to a relation type
- two foreign keys, one for each table
- the primary key of the relation type is the combination of the foreign keys
- minimal cardinalities cannot be enforced

Example



- Client(**client_id**)
- Account(**account_id**)
- Uses(**client_id, account_id**) both non nullable

Outline

SQL

What is SQL?

- SQL is relational data management language
- Structured Query Language pronounced sequel
- developed initially by IBM as an "implementation" of the relational model
- SQL is a standard since 1986 (numerous versions)

Implementations

- SQL is “supported” by all relational database management systems
- many open source solutions (MySQL/MariaDB, PostgreSQL, SQLite, etc.)
- but many variations in the support level (portability is not guaranteed)

SQL Components

Multiple aspects

- Data Definition/Description Language
  - relational model description
  - domain definition
- Data Manipulation Language: insertion, suppression and modification
- Data Query Language
  - read only manipulation
  - selection, filtering, grouping, etc.
- Data Control Language
  - access control to the databases
  - users, roles, permissions, etc.

SQL Data Description Language

Relation

- a relation (type) is created in SQL by

```
CREATE TABLE relation_name (column_name domain, ...);
```

- SQL supports numerous default domains (implementation dependent!):
  - `INT`: integer
  - `DATETIME`, `DATE` and `TIME`: date and time
  - `BOOLEAN`: true or false
  - `CHAR(n)` and `VARCHAR(n)`: string with maximum size n
- constraints can be specified after the domain:
  - **`NOT NULL`**: non nullable
  - **`PRIMARY KEY`**: self explanatory
  - **`UNIQUE`**: candidate key
  - **`DEFAULT`**: used to specify a default value

Example

Relational model to SQL

Client(**client_id**, first_name, last_name, birth_date)

```
CREATE TABLE Client (
  client_id INT PRIMARY KEY,
  first_name VARCHAR(30) NOT NULL,
  last_name VARCHAR(30) NOT NULL,
  birth_date DATETIME NOT NULL
);
```

Domains

SQL domains

- domains can be created in SQL
- typical form

```
CREATE DOMAIN Gender AS VARCHAR(6)
  CHECK (VALUE IN ('Female','Male'));
```

- unsupported in many implementations (e.g. MySQL)

Constraints based version

- constraints can be added to the table creation
- **CHECK** can be used to implemented domains
- less elegant (no centralized definition)

Example

Relational model to SQL

Client(**client_id**, gender, first_name, last_name, birth_date)

```
CREATE TABLE Client (
  client_id INT PRIMARY KEY,
  gender AS VARCHAR(6),
  CONSTRAINT gc CHECK(gender IN ('Female','Male')),
  first_name VARCHAR(30) NOT NULL,
  last_name VARCHAR(30) NOT NULL,
  birth_date DATETIME NOT NULL
);
```

Keys

Primary keys

- primary keys are not mandatory in SQL
- but they should be specified!
- `UNIQUE` is useful as a constraint
- a primary key can be made with several columns using
  `PRIMARY KEY (COL1, COL2, ...)`
  in the table creation

Foreign keys

- declared as `FOREIGN KEY (column)` during table creation
- together with a `REFERENCES table(column)`
- a foreign key can be a set of columns

Example

Relational model to SQL

- Client(**client_id**)
- Account(**account_id**,client_id)

```
CREATE TABLE Client(client_id int PRIMARY KEY);
CREATE TABLE Account(account_id int PRIMARY KEY,
   client_id int NOT NULL,
   FOREIGN KEY(client_id) REFERENCES Client(client_id));
```

Referential integrity

Foreign keys

- must reference an existing primary key
- SQL allows one to handle consequences of tuple modifications
  - what happens if the primary key of a tuple is modified?
  - what happens if a tuple is deleted?
- `ON DELETE something` and `ON UPDATE something`
- with something being
  - `CASCADE`: propagate the modification to referring tuples
  - `RESTRICT`: forbid the modification if there are referring tuples
  - `SET NULL` or `SET DEFAULT`: modify the foreign key in the referring tuples as described

Example

Relational model to SQL

- Client(**client_id**)
- Account(**account_id**,client_id)

```
CREATE TABLE Client(client_id int PRIMARY KEY);
CREATE TABLE Account(account_id int PRIMARY KEY,
   client_id int NOT NULL,
   FOREIGN KEY(client_id) REFERENCES Client(client_id))
   ON DELETE RESTRICT ON UPDATE CASCADE;
```

Additional commands

Modifying the model

- **`DROP TABLE`** `name`: delete a table
- **`ALTER TABLE`** `...`:
  - data model modification
  - long list of possibilities
    - adding/removing a column
    - changing the properties of a column (domain, constraints, etc.)
    - etc.

SQL Data Query Language

The `SELECT` command

- the main query command in SQL
- general form

```
SELECT something FROM somewhere
[WHERE conditions] [GROUP BY grouping]
[HAVING group conditions] [ORDER BY something]
```

- provides all the manipulations available in R+dyplr or Python+pandas:
  - subsetting, filtering, transforming, ordering
  - summarizing
  - joining

Subsetting

Subsetting

- simple `SELECT` queries can be used to subset a relation on interesting columns
- general form `SELECT col1, ..., col2 FROM table;`

Exemple

```
SELECT client_id, gender FROM Client;
```

Client relation

client_id	gender	birth_date	district_id
1	F	1970-12-13	18
2	M	1945-02-04	1
3	F	1940-10-09	1
4	M	1956-12-01	5
5	F	1960-07-03	5

Result relation

client_id	gender
1	F
2	M
3	F
4	M
5	F

Transformation

Expression and renaming

- columns may be renamed using `orig_name AS new_name` in the **`SELECT`** command
- simple calculations may also be performed on columns including the results as new columns

Example

```
SELECT A2 AS Name, A4 AS Population,
       A5+A6+A7+A8 AS Municipalities FROM District;
```

District relation

district_id	A2	A3	A4	A5	A6	A7	A8
1.00	Hl.m. Praha	Prague	1204953.00	0.00	0.00	0.00	1.00
2.00	Benesov	central Bohemia	88884.00	80.00	26.00	6.00	2.00
3.00	Beroun	central Bohemia	75232.00	55.00	26.00	4.00	1.00
4.00	Kladno	central Bohemia	149893.00	63.00	29.00	6.00	2.00
5.00	Kolin	central Bohemia	95616.00	65.00	30.00	4.00	1.00

Transformation

Expression and renaming

- columns may be renamed using `orig_name AS new_name` in the `SELECT` command
- simple calculations may also be performed on columns including the results as new columns

Example

```
SELECT A2 AS Name, A4 AS Population,
       A5+A6+A7+A8 AS Municipalities FROM District;
```

Result relation

Name	Population	Municipalities
Hl.m. Praha	1204953.00	1.00
Benesov	88884.00	114.00
Beroun	75232.00	86.00
Kladno	149893.00	100.00
Kolin	95616.00	100.00

Filtering

Selecting tuples

- the **WHERE** clause can be used to select tuples fulfilling some conditions
- general form

```
SELECT columns FROM table WHERE conditions;
```

Exemple

```
SELECT client_id, birth_date FROM Client
       WHERE Gender='F';
```

Client relation

client_id	gender	birth_date	district_id
1	F	1970-12-13	18
2	M	1945-02-04	1
3	F	1940-10-09	1
4	M	1956-12-01	5
5	F	1960-07-03	5

Result relation

client_id	birth_date
1	1970-12-13
3	1940-10-09
5	1960-07-03
8	1938-02-21
11	1950-08-22

Aggregation

Global summaries

- aggregation functions can be used in the result part of the **SELECT** command
- they operate at the column level
- some examples:
  - **COUNT** and **COUNT**(**DISTINCT**(.))
  - **MAX**, **MIN**, **SUM**
  - **AVG**, STD, VARIANCE

Financial database

```
SELECT COUNT(birth_date) FROM Client WHERE Gender='F';
```

gives 2645, while

```
SELECT COUNT(DISTINCT(birth_date)) FROM Client
       WHERE Gender='F';
```

gives 2461.

Conditional aggregation

Split apply combine in SQL

- the **`GROUP BY`** clause of the **`SELECT`** command provides conditional analysis
- it *splits* the relation into groups of tuples on which it *applies* chosen aggregation functions
- groups can be further selected based on global properties with the **`HAVING`** clause

General form

```
SELECT aggregates FROM relation
       [WHERE conditions]
       GROUP BY columns
       [HAVING group conditions]
```

Split apply combine

X	Y
2	A
1	B
4	B
3	B
1	B
2	C
1	A
3	C
3	A
2	A
2	A
3	A
3	B
1	A
1	A

Split

Y	X
A	2
A	1
A	3
A	2
A	2
A	3
A	1
A	1

Y	X
B	1
B	4
B	3
B	1
B	3

Y	X
C	2
C	3

Apply

Y	Sum
A	15

Y	Sum
B	12

Y	Sum
C	5

Combine

Y	Sum
A	15
B	12
C	5

Examples

Count clients per gender

```
SELECT gender, COUNT(gender) AS number FROM Client
       GROUP BY gender;
```

gender	number
F	2645
M	2724

Count clients per district id and gender

```
SELECT district_id, gender, COUNT(gender) AS number
       FROM Client GROUP BY district_id, gender;
```

district_id	gender	number
1	F	324
1	M	339
2	F	26
2	M	20
3	F	27

Example

Count clients per district id and gender

and keep only groups with more than 50 persons;

```
SELECT district_id, gender, COUNT(gender) AS number
       FROM Client GROUP BY district_id, gender
       HAVING COUNT(*)>=50;
```

district_id	gender	number
1	F	324
1	M	339
54	F	75
54	M	80
64	F	57
64	M	52
70	F	88
70	M	81
72	F	50
72	M	54
74	F	84
74	M	96

Joins

Joining relations

- a **`SELECT`** command can operate on multiple relations at once (in the **`FROM`** clause)
- default behavior: cartesian product (a.k.a. a cross join)
- key based joins are explicitly constructed via **`WHERE`** or **`ON`** conditions
  - dotted notation: `relation.`**`column`**
  - alias for relation in the **`FROM`** clause
  - support for complex conditions (not only key equality)
  - inner joins by default if implicit
- implicit joins: **`WHERE`**
- explicit joins: **`JOIN ... ON`**

Example

Financial data

- goal: gender distribution per region
- sources:
  - genders in the Client table
  - regions in the District table
  - district_id is a foreign key in the Client table which references to the District table
- first step (join)

```
SELECT gender, A3 AS region From Client, District
       WHERE Client.district_id = District.district_id;
```

- full query (with the group by)

```
SELECT C.gender, D.A3 AS region, count(*) as number
       FROM Client C, District D
       WHERE C.district_id = D.district_id
       GROUP BY region, C.gender;
```

Example

Financial data

- goal: gender distribution per region
- sources:
  - genders in the Client table
  - regions in the District table
  - district_id is a foreign key in the Client table which references to the District table
- first step (join)

```
SELECT gender, A3 AS region From Client
       INNER JOIN District
       ON Client.district_id = District.district_id;
```

- full query (with the group by)

```
SELECT C.gender, D.A3 AS region, count(*) as number
      FROM Client C INNER JOIN District D
      WHERE C.district_id = D.district_id
      GROUP BY region, C.gender;
```

Example

District relation

district_id	A2	A3
1.00	Hl.m. Praha	Prague
2.00	Benesov	central Bohemia
3.00	Beroun	central Bohemia
4.00	Kladno	central Bohemia
5.00	Kolin	central Bohemia

Client relation

client_id	gender	district_id
1	F	18
2	M	1
3	F	1
4	M	5
5	F	5

Joined relations

gender	region
F	south Bohemia
M	Prague
F	Prague
M	central Bohemia
F	central Bohemia

Results

region	gender	Number
central Bohemia	F	336
central Bohemia	M	328
east Bohemia	F	318
east Bohemia	M	342
north Bohemia	F	281
north Bohemia	M	280
north Moravia	F	437
north Moravia	M	483
Prague	F	324
Prague	M	339
south Bohemia	F	231
south Bohemia	M	218
south Moravia	F	474
south Moravia	M	463
west Bohemia	F	244
west Bohemia	M	271

Example

Financial data

- goal: persons with an account in a bank another district than their home district
- sources:
  - home district in the Client table
  - bank district in the Account table
  - links between client and account in the Disposition table
- query

```
SELECT C.client_id, C.district_id AS home,
       A.district_id AS bank
       FROM Client C INNER JOIN Disposition D
            INNER JOIN Account A
       WHERE C.client_id = D.client_id
             AND D.account_id=A.account_id
             AND C.district_id != A.district_id;
```

Example

Results

client_id	home	bank
12	40	70
20	46	70
42	68	67
44	38	1
47	76	36
48	16	64
77	1	48
98	33	26
99	33	26
102	74	54
103	74	54
114	74	68
115	74	68
138	54	68
151	51	5
153	13	2
162	65	75
193	1	23
195	74	60
196	74	60

Outer joins

Similar approach

- replace `INNER JOIN` by
  - `LEFT OUTER JOIN` (or `LEFT JOIN`)
  - `RIGHT OUTER JOIN` (or `RIGHT JOIN`)
  - `FULL OUTER JOIN` (or `FULL JOIN`)

Other possibilities

Not described here

- `ORDER BY`: results ordering
- `DISTINCT`: unique results only
- nested queries: `SELECT` queries in other `SELECT` queries
- set operations such as `UNION` and `INTERSECT`

SQL Data Manipulation Language

INSERT

- inserting a tuple into a relation:
  ```
  INSERT INTO table VALUES (...);
  ```
- variants include
  ```
  INSERT INTO table (columns...) VALUES (...);
  ```
  to specify the column names (`NULL` is assigned to missing columns)

DELETE

- deleting is done conditionally, using a `WHERE` clause
- general syntax
  ```
  DELETE FROM table WHERE condition;
  ```

Updating

UPDATE

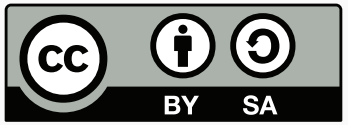
- used to alter tuples
- general syntax

```
UPDATE table
  SET column = value [,column = value...]
  [WHERE condition];
```

Changelog

- March 2020
  - fixed typos
  - added typical domains
- December 2019: added a short introduction to SQL
- November 2019: initial version

Licence



Version

Last git commit: 2021-01-19
By: Fabrice Rossi (Fabrice.Rossi@apiacoa.org)
Git hash: 97cfd0a9975cf193f5790845c00e476c1572a327