University of Zurich[UZH]

GEO 874 | HS25
University of Zürich

2. Lecture Introduction to Databases

SQL – the Language for Relational Databases

Esra Suel
Dept. of Geography, University of Zürich

Rolf Meile
Eidg. Forschungsanstalt für Wald, Schnee und Landschaft (WSL)

kudos to Dr. Zhiyong Zhou, Dr. Cheng Fu, Dr. Haosheng Huang for providing this document

Recap Lecture 1

- „Nutty Nuggets“ Example
- The database characteristics – „DB in a nutshell“
- Datatabase management systems
- The relational model
- Pros- and cons- of database use
- Products and big players

Learning objectives L2

- ✓ Gain a first overview of Structured Query Language (SQL), the standard language for relational DBMS.
- ✓ You will learn the basic SQL language for:
  - ✓ **Querying** tables,
  - ✓ **Modification** of tables,
  - ✓ **Modification** of data in the tables.
- ✓ You will understand the role of **primary and foreign keys**
- ✓ You will learn about **views** and will be able to create them.
- ✓ You will be able to link multiple tables through **joins**.

Individual initiative!

- This lecture only provides an overview of the basics of SQL!
- For the exercises, you will need to build SQL queries on your own! Check the **Resources**
  - http://www.geo.uzh.ch/microsite/geo874/
  - Ramez A. Elmasri / Shamkant B. Navathe (2014), **Ch4:** Basic SQL, **Ch5:** More SQL, 82-147.

- SQL day in three parts
  - Part 1: Lecture
  - Part 2: Demo [later, during lab]
  - Part 3: Exercise (self-correct) [later, during lab]

SQL – the language for relational databases

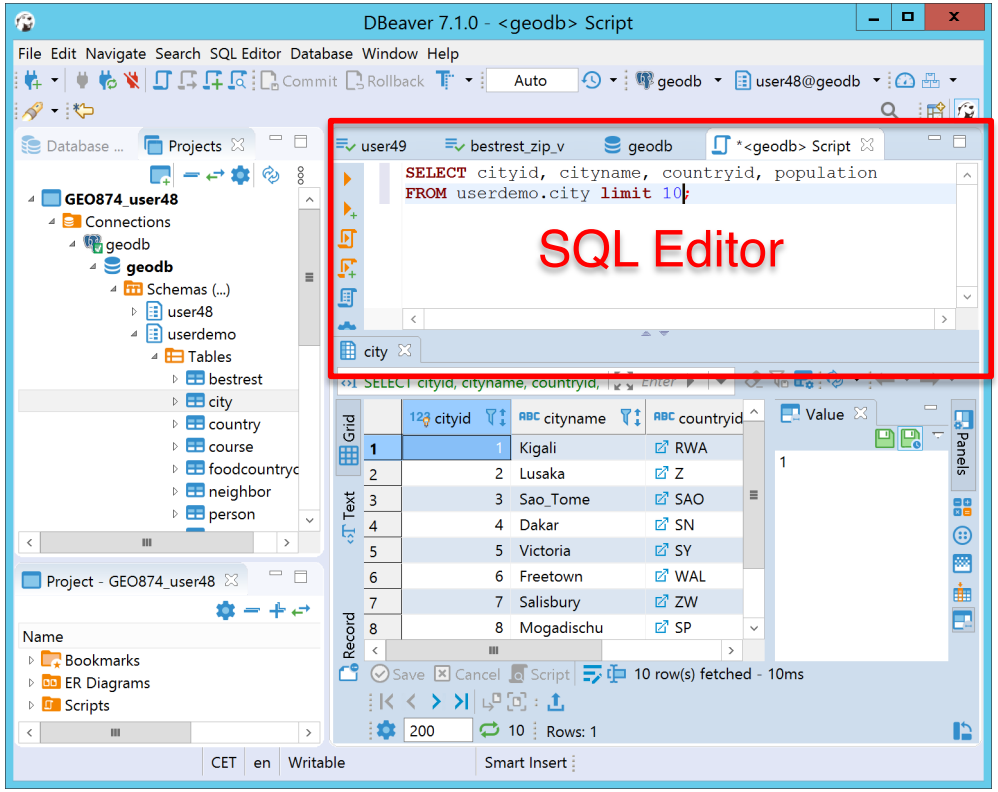
1. **SQL-Intro**
2. Data types
3. SQL concepts: database, schema, table
4. Data Definition Language (DDL)
5. Data Manipulation Language (DML)
6. Data Query Language (DQL)
7. Definition of Views

SQL-Intro

Motivation

Structured Query Language (SQL) in relational DB allows:

1. Creating tables (*relations*)
2. Loading data into tables
3. Modifying data in the tables
4. Querying data in the tables
5. Deleting data
6. Modifying table structures
7. Deleting tables
8. Management of access and data protection



Characteristics of SQL

- **Ad hoc** – a statement, not a full program
- **Descriptive** – what I want, not how to get it
- **Set oriented** - Work on collections of data records
- **Isolation** – the result of a query is complete on its own
  - and can also serve as input for any other queries (nested queries)
- **Efficiency** – efficient execution of operations – optimized by the SQL engine (query optimizer in DBMS)

Expanded in: https://en.wikipedia.org/wiki/Codd%27s_12_rules

SQL - Overview

- SQL (/ˈɛs kjuː ˈɛl/ or /ˈsiːkwəl/)
- has a mathematical foundation
- Standard (ANSI/ISO SQL)...
- But rarely implemented entirely and often enriched with additions (oracle dialect, MySQL dialect, PostgreSQL dialect, ...)
- The SQL standard continuously refined

Year	Official standard	Informal name	Comments
1986 1987	ANSI X3.135:1986 ISO/IEC 9075:1987 FIPS PUB 127	SQL-86 SQL-87	First formalized by ANSI, adopted as FIPS PUB 127
1989	ANSI X3.135-1989 ISO/IEC 9075:1989 FIPS PUB 127-1	SQL-89	Minor revision that added integrity constraints, adopted as FIPS PUB 127-1
1992	ANSI X3.135-1992 ISO/IEC 9075:1992 FIPS PUB 127-2	SQL-92 SQL2	Major revision (ISO 9075), *Entry Level* SQL-92, adopted as FIPS PUB 127-2
1999	ISO/IEC 9075:1999	SQL:1999 SQL3	Added regular expression matching, recursive queries (e.g., transitive closure), triggers, support for procedural and control-of-flow statements, nonscalar types (arrays), and some object-oriented features (e.g., structured types), support for embedding SQL in Java (SQL/OLB) and vice versa (SQL/JRT)
2003	ISO/IEC 9075:2003	SQL:2003	Introduced XML-related features (SQL/XML), window functions, standardized sequences, and columns with autogenerated values (including identity columns)
2006	ISO/IEC 9075-14:2006	SQL:2006	Adds Part 14, defines ways that SQL can be used with XML. It defines ways of importing and storing XML data in an SQL database, manipulating it within the database, and publishing both XML and conventional SQL data in XML form. In addition, it lets applications integrate queries into their SQL code with XQuery, the XML Query Language published by the World Wide Web Consortium (W3C), to concurrently access ordinary SQL-data and XML documents.[32]
2008	ISO/IEC 9075:2008	SQL:2008	Legalizes ORDER BY outside cursor definitions. Adds INSTEAD OF triggers, TRUNCATE statement,[33] FETCH clause
2011	ISO/IEC 9075:2011	SQL:2011	Adds temporal data (PERIOD FOR)[34] (more information at Temporal ... enhancements for window functions and FETCH clause.[35]
2016	ISO/IEC 9075:2016	SQL:2016	Adds row pattern matching, polymorphic table functions, operations on JSON data stored in character string fields
2019	ISO/IEC 9075-15:2019	SQL:2019	Adds Part 15, multidimensional arrays (MDarray type and operators)
2023	ISO/IEC 9075:2023	SQL:2023	Adds data type JSON (SQL/Foundation); Adds Part 16, Property Graph Queries (SQL/PGQ)

We focus on principles, not implementations

BUT

We use PostgreSQL in this course – some actual SQL code is shown and may differ in other DBMS.

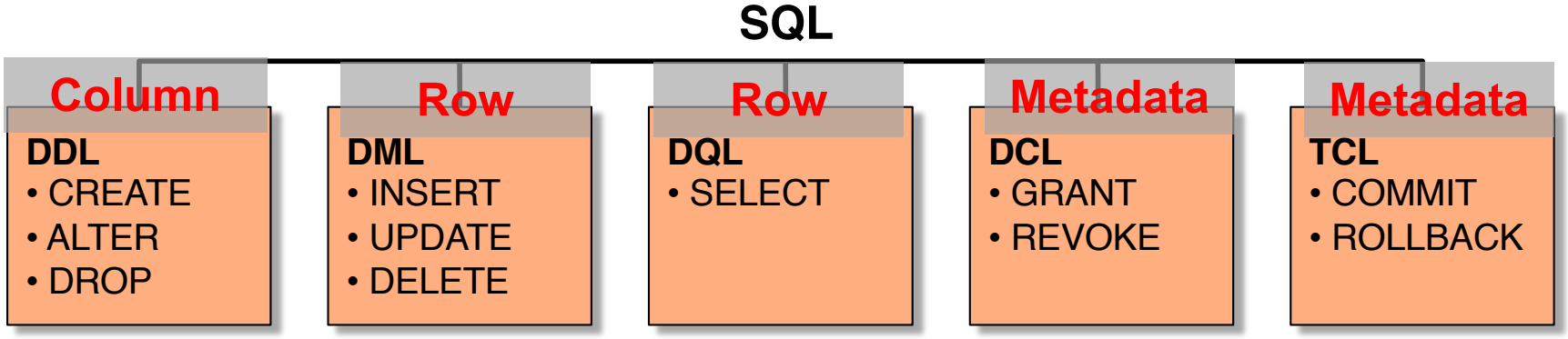
➲ http://en.wikipedia.org/wiki/SQL

SQL is a comprehensive database language

DDL: Data Definition Language – defines the structure of the database.
DML: Data Manipulation Language – modifies data in the database
DQL: Data Query Language – retrieves data from the database
DCL: Data Control Language – manages user permissions and access
TCL: Transaction Control Language – handles transactions & ensures data integrity

SQL:

- Provides pre-defined data types
- Allows the definition of Schemas & Views
- Handling of rights of access



SQL – the language for relational databases

1. SQL-Intro
2. **Data types**
3. SQL concepts: database, schema, table
4. Data Definition Language (DDL)
5. Data Manipulation Language (DML)
6. Data Query Language (DQL)
7. Definition of Views

SQL General Data types

Data type	Description
CHARACTER(n)	Character string. Fixed-length n
VARCHAR(n) or CHARACTER VARYING(n)	Character string. Variable length. Maximum length n
BINARY(n)	Binary string. Fixed-length n
BOOLEAN	Stores TRUE or FALSE values
VARBINARY(n) or BINARY VARYING(n)	Binary string. Variable length. Maximum length n
INTEGER(p)	Integer numerical (no decimal). Precision p
SMALLINT	Integer numerical (no decimal). Precision 5
INTEGER	Integer numerical (no decimal). Precision 10
BIGINT	Integer numerical (no decimal). Precision 19
DECIMAL(p,s)	Exact numerical, precision p, scale s. Example: decimal(5,2) is a number that has 3 digits before the decimal and 2 digits after the decimal
NUMERIC(p,s)	Exact numerical, precision p, scale s. (Same as DECIMAL)
FLOAT(p)	Approximate numerical, mantissa precision p. A floating number in base 10 exponential notation. The size argument for this type consists of a single number specifying the minimum precision
REAL	Approximate numerical, mantissa precision 7
FLOAT	Approximate numerical, mantissa precision 16
DOUBLE PRECISION	Approximate numerical, mantissa precision 16
DATE	Stores year, month, and day values
TIME	Stores hour, minute, and second values
TIMESTAMP	Stores year, month, day, hour, minute, and second values
INTERVAL	Composed of a number of integer fields, representing a period of time, depending on the type of interval
ARRAY	A set-length and ordered collection of elements
MULTISET	A variable-length and unordered collection of elements
XML	Stores XML data

However, different DBMS offer different choices for the data type definition.

- Same data types might have different names in different DBMS.
- And even if the name is the same, the size and other details may be different!
- Always check the documentation!

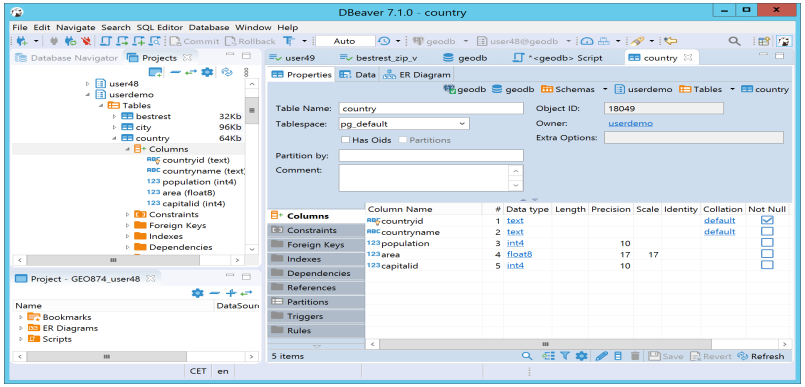
http://www.w3schools.com/sql/sql_datatypes_general.asp

Comparison of SQL Data type implementations

Data type	Access	SQLServer	Oracle	MySQL	PostgreSQL
boolean	Yes/No	Bit	Byte	N/A	Boolean
integer	Number (integer)	Int	Number	Int Integer	Int Integer
float	Number (single)	Float Real	Number	Float	Numeric
currency	Currency	Money	N/A	N/A	Money
string (fixed)	N/A	Char	Char	Char	Char
string (variable)	Text (<256) Memo (65k+)	Varchar	Varchar Varchar2	Varchar	Varchar
binary object	OLE Object Memo	Binary (fixed up to 8K) Varbinary (<8K) Image (<2GB)	Long Raw Blob	Blob Text	Binary Varbinary

Beware!

http://www.w3schools.com/sql/sql_datatypes_general.asp



PostgreSQL SQL-Data types

Name	Aliases	Description
bigint	int8	signed eight-byte integer
bigserial	serial8	autoincrementing eight-byte integer
bit [(*n*)]		fixed-length bit string
bit varying [(*n*)]	varbit	variable-length bit string
boolean	bool	logical Boolean (true/false)
box		rectangular box on a plane
bytea		binary data ("byte array")
character [(*n*)]	char [(*n*)]	fixed-length character string
character varying [(*n*)]	varchar [(*n*)]	variable-length character string
cidr		IPv4 or IPv6 network address
circle		circle on a plane
date		calendar date (year, month, day)
double precision	float8	double precision floating-point number (8 bytes)
inet		IPv4 or IPv6 host address
integer	int, int4	signed four-byte integer
interval [*fields*] [(*p*)]		time span
json		textual JSON data
jsonb		binary JSON data, decomposed
line		infinite line on a plane
lseg		line segment on a plane
macaddr		MAC (Media Access Control) address
money		currency amount
numeric [(*p*, *s*)]	decimal [(*p*, *s*)]	exact numeric of selectable precision
path		geometric path on a plane
pg_lsn		PostgreSQL Log Sequence Number
point		geometric point on a plane
polygon		closed geometric path on a plane
real	float4	single precision floating-point number (4 bytes)
smallint	int2	signed two-byte integer
smallserial	serial2	autoincrementing two-byte integer
serial	serial4	autoincrementing four-byte integer
text		variable-length character string
time [(*p*)] [without time zone]		time of day (no time zone)
time [(*p*)] with time zone	timetz	time of day, including time zone
timestamp [(*p*)] [without time zone]		date and time (no time zone)
timestamp [(*p*)] with time zone	timestamptz	date and time, including time zone
tsquery		text search query
tsvector		text search document
txid_snapshot		user-level transaction ID snapshot
uuid		universally unique identifier
xml		XML data

The following types (or spellings thereof) are specified by SQL:
bigint, bit, bit varying, boolean, char, character varying, character, varchar, date, double precision, integer, interval, numeric, decimal, real, smallint, time (with or without time zone), timestamp (with or without time zone), xml.

Users can add new types in PostgreSQL using the CREATE TYPE command.

https://www.postgresql.org/docs/12/datatype.html

Selected types: Numeric

integer, smallint, bigint integer (number without decimal point), with 4, 2, and 8 bytes

numeric(p, s) numbers, opt. can also specify a precision (total number of digits) and scale (number of digits to the right of the decimal point) E.g., `1234567.123 is a numeric(10,3)`

numeric(p) integer with *p* digits (having 0 scale), e.g., `12345 is a numeric(5)`

The types decimal and numeric are equivalent

real 4 bytes, variable-precision numeric types

double precision 8 bytes, variable-precision numeric types

Smallserial, serial, bigserial autoincrementing integer

Name	Storage Size	Description	Range
smallint	2 bytes	small-range integer	-32768 to +32767
integer	4 bytes	typical choice for integer	-2147483648 to +2147483647
bigint	8 bytes	large-range integer	-9223372036854775808 to +9223372036854775807
decimal	variable	user-specified precision, exact	up to 131072 digits before the decimal point; up to 16383 digits after the decimal point
numeric	variable	user-specified precision, exact	up to 131072 digits before the decimal point; up to 16383 digits after the decimal point
real	4 bytes	variable-precision, inexact	6 decimal digits precision
double precision	8 bytes	variable-precision, inexact	15 decimal digits precision
smallserial	2 bytes	small autoincrementing integer	1 to 32767
serial	4 bytes	autoincrementing integer	1 to 2147483647
bigserial	8 bytes	large autoincrementing integer	1 to 9223372036854775807

Selected (PostgreSQL) SQL-Data types

Character types (e.g., string, text)

Name	Description
character varying(*n*), varchar(*n*)	variable-length with limit
character(*n*), char(*n*)	fixed-length, blank padded
text	variable unlimited length

Date/time types

Name	Storage Size	Description	Low Value	High Value	Resolution
timestamp [(*p*)] [without time zone]	8 bytes	both date and time (no time zone)	4713 BC	294276 AD	1 microsecond / 14 digits
timestamp [(*p*)] with time zone	8 bytes	both date and time, with time zone	4713 BC	294276 AD	1 microsecond / 14 digits
date	4 bytes	date (no time of day)	4713 BC	5874897 AD	1 day
time [(*p*)] [without time zone]	8 bytes	time of day (no date)	00:00:00	24:00:00	1 microsecond / 14 digits
time [(*p*)] with time zone	12 bytes	times of day only, with time zone	00:00:00+1459	24:00:00-1459	1 microsecond / 14 digits
interval [*fields*] [(*p*)]	16 bytes	time interval	-178000000 years	178000000 years	1 microsecond / 14 digits

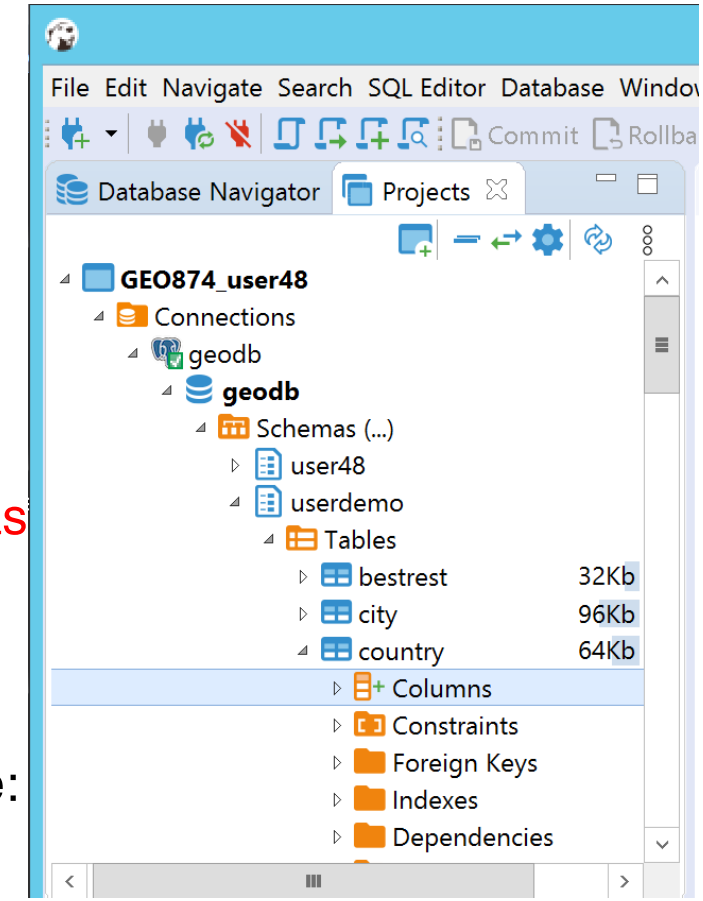
SQL – the language for relational databases

1. SQL-Intro
2. Data types
3. **SQL concepts: database, schema, table**
4. Data Definition Language (DDL)
5. Data Manipulation Language (DML)
6. Data Query Language (DQL)
7. Definition of Views

Database, Schema, Table

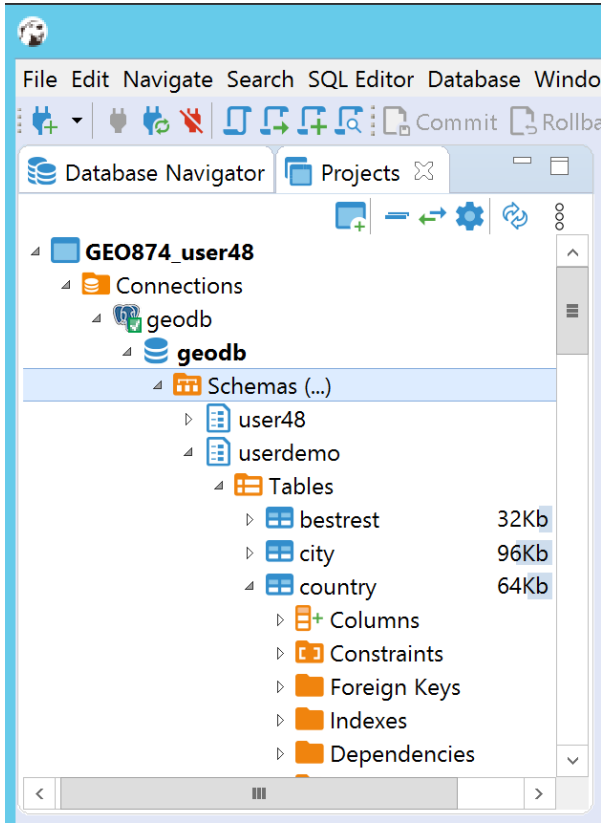
- Database structure and organization
  - How data is organized in a database (e.g., divided into DB tables in case of relational DBs)

- PostgreSQL
  - Server instance (or database cluster): sc02.geo.uzh.ch
  - A server instance can have one or more named databases (e.g., „geodb“).
  - A database contains one or more named schemas (= namespaces), which in turn contain tables.
  - Schemas also contain other kinds of named objects, e.g., views, data types, functions, operations.
  - Three default schemas when creating a database:
    - Public: your own tables go unless you specify otherwise
    - pg_catalog: contains the system tables and all the built-in data types, functions, and operators
    - information_schema: standards-conformant „pg_catalog“



Database, Schema, Table

- Why using schemas:
  - To allow many users to use one DB without interfering with each other (e.g., each of you have an individual schema),
  - To organize DB objects (e.g., tables) into logical groups to make them more manageable.
  - The same object name can be used in different schemas without conflict
- In our lab, we have a database named „geodb“, and each student and lecturer is allocated a schema (e.g., „user01“)



- To create or access a table within a schema

```
SELECT * FROM user01.mytable
```

```
SELECT * FROM geodb.user01.mytable
```

SQL – the language for relational databases

1. SQL-Intro
2. Datatypes
3. SQL concepts: database, schema, table
4. **Data Definition Language (DDL)**
5. Data Manipulation Language (DML)
6. Data Query Language (DQL)
7. Definition of Views

Table Relation

The terms **Table** and **Relation** are often used interchangeably.

- A table: a collection of records (rows)
- Order of records within a table is irrelevant.
- SQL terminology: *table*, *row*, *column* relate to the concepts of *Relation*, *Tuple*, and *Attribute*

- In relational DB models data are organised in tabular *relations*
- Sets of Tuples (un-ordered)

- Term often used during conceptual DB design

- Table names often do not contain space, e.g., „~~customer information~~“

a column/field/attribute

Table: customers

ID	Name	Company	Email
1001	Vedat Diker	UMD	rwe@umd.com
1002	Bugs Bunny	AMK, Inc.	bugs@amk.com
1003	Will Coyo	AMK, Inc.	will@amk.com

a record/row/tuple

DDL- Data Definition Language

Operations to Create/Delete and Modify **tables**

- CREATE
- ALTER
- DROP

Employee

Column_name	**Type**
Surname	`varchar(25)`
Name	`varchar(25)`

Creation of tables

CREATE TABLE – creation of a new table

- Table Name (in the format of schema_name.table_name)
- Attributes & Data types
- Constraints

```
CREATE TABLE user01.Person (
        PersNr              numeric(7) PRIMARY KEY,
        Name                varchar(25) NOT NULL,
        FirstName           varchar(25) NOT NULL,
        DepartmentID        numeric(6) NOT NULL,
        Salary              numeric(7,2) CHECK (Salary >0),
        ZipCode             numeric(4)
)
```

Constraints

- Define **integrity restrictions** on the **values** of one or more **attributes** and **apply** to entire **columns** or even **tables**.
  - If there is any violation between the constraint and the data action, the action is stopped by the constraint.
  - E.g., In the constraint „**`CHECK`** `(Salary >0)`“, inserting a new record with salary as 0 will fail.

- Constraints can be specified when the table is created (inside the CREATE TABLE statement):
  - Column-level definition: The constraints can be specified immediately after the column definition.
  - Table-level definition: The constraints can be specified after all the columns are defined
- Or after the table is created (inside the ALTER TABLE statement)

- Constraints available in PostgreSQL: not null, default, unique, check, primary key, foreign key

Constraints: NOT NULL

- By default, a table column can hold NULL values
- The NOT NULL constraint enforces a column to NOT accept NULL values.
  - It enforces a field to always contain a value.
- A not-null constraint is always written as a column constraint (immediately after the column definition).

```
CREATE TABLE user01.Person (
       PersNr        numeric(7) NOT NULL,
       Name          varchar(25) NOT NULL,
       FirstName     varchar(25) NOT NULL
)
```

Constraints: DEFAULT

- The DEFAULT constraint is used to insert a default value into a column.

- The default value will be added to all new records, if no other value is specified.

```
CREATE TABLE user01.Person (
       PersNr          numeric(7) NOT NULL,
       Name            varchar(25) NOT NULL,
       FirstName       varchar(25) NOT NULL,
       DepartID        numeric(5) DEFAULT 12345
)
```

Constraints: UNIQUE

- Unique constraints ensure that the data contained in a column, or a group of columns, is unique among all the rows in the table.
  - Valid for a column or combination of columns. A single entry value is unique in the entire column or combination of columns
  - multiple UNIQUE constraints are possible in a table
  - Column-level definition

```
CREATE TABLE user01.Person (
         PersNr            numeric(7) NOT NULL UNIQUE,
         Name              varchar(25) NOT NULL,
         FirstName         varchar(25) NOT NULL
)
```

  - Table-level definition (a unique constraint for a group of columns)

```
CREATE TABLE user01.grades (
         StudentID         numeric(7) NOT NULL,
         CourseID          numeric(7) NOT NULL,
         Grade             numeric(3) NOT NULL,
         UNIQUE (StudentID, CourseID)
)
```

Constraints: CHECK

- The CHECK constraint is used to limit the value range that can be placed in a column.
- If you define a CHECK constraint on a single column it allows only certain values for this column.
- If you define a CHECK constraint on a table it can limit the values in certain columns based on values in other columns in the row.
  - Column-level definition

```
CREATE TABLE user01.student(
        StudentID       numeric(7) NOT NULL UNIQUE,
        Name            varchar(50) NOT NULL,
        BirthYear       numeric(4) NOT NULL CHECK (BirthYear<2005)
)
```

  - Table-level definition

```
CREATE TABLE user01.student(
        StudentID       numeric(7) NOT NULL UNIQUE,
        Name            varchar(50) NOT NULL,
        BirthYear       numeric(4) NOT NULL,
        Gender          char (1),
        CHECK (BirthYear<2005 AND gender in ('M', 'F'))
)
```

Key attributes

- Some Attributes can serve as KEYs
- **Primary key**: defines a column or combination of columns that uniquely identifies each row in the table (i.e., having UNIQUE and NOT NULL)
  - There may be multiple candidates for PK, only one is selected.
  - A PK might contain one or more columns (e.g., combinations of StudentID, CourseID for the grades table)
  - Theoretically, a relation does not have to have a PK – but better to have one ...
- Examples of PKs: Postcode, AHV Number, Passport Number
  - may have semantics attached

Ort

PLZ	Location
6330	Cham
9000	St. Gallen
8002	Zürich
6300	Zug
8400	Winterthur
8003	Zürich
3000	Bern
8001	Zürich
8051	Zürich

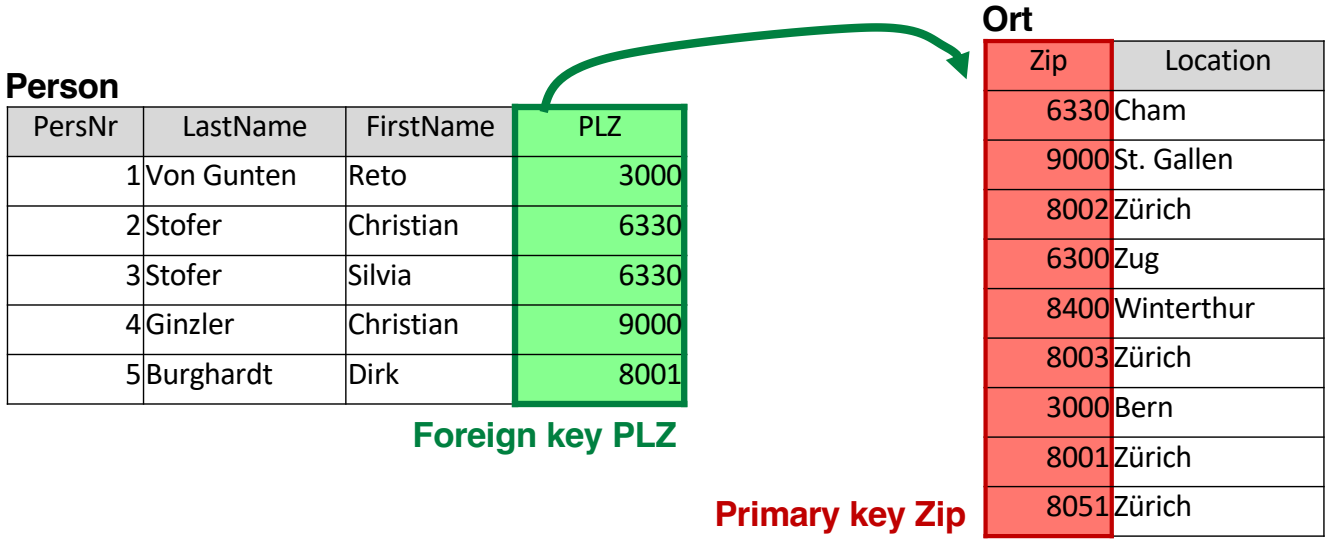
```
CREATE TABLE user01.Person (
          PersNr              numeric(7) PRIMARY KEY,
          Name                varchar(25) NOT NULL,
          FirstName           varchar(25) NOT NULL
)
```

```
CREATE TABLE user01.grades (
          StudentID           numeric(7),
          CourseID            numeric(7),
          Grade               numeric(3) NOT NULL,
          PRIMARY KEY(StudentID, CourseID)
)
```

Can StudentID or CourseID take NULL values? **No!**

Key attributes

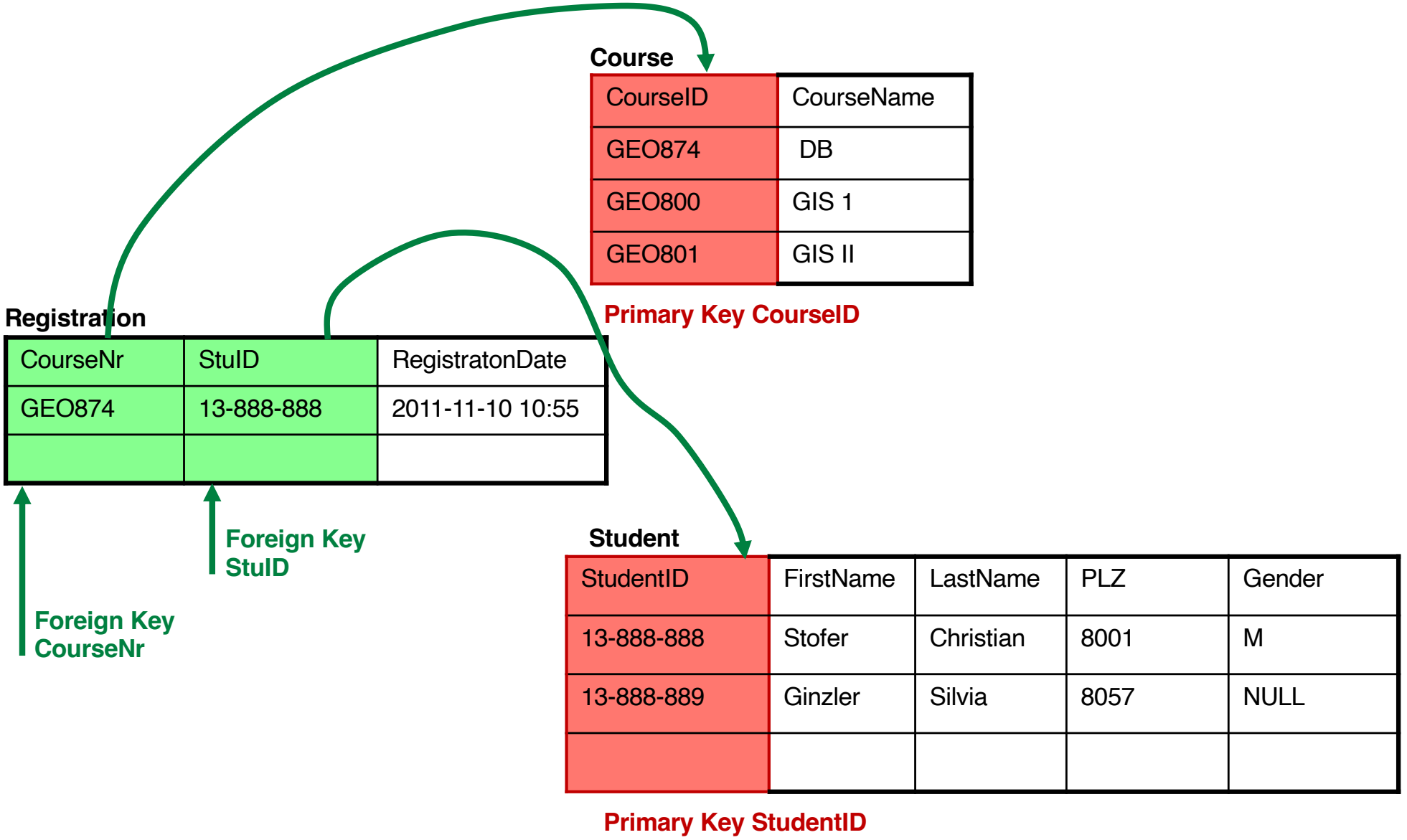
- **Foreign key**: refers to the value of a primary key in another table.
  - Values: only from the set of values of the primary key in the referenced relation
  - Referential integrity: Must be present in the referenced table




```
CREATE TABLE user01.Ort(
    Zip         numeric(4)  PRIMARY KEY,
    Location    varchar(25) NOT NULL
)
```

```
CREATE TABLE user01.Person (
    PersNr    numeric(7)  PRIMARY KEY,
    LastName  varchar(25) NOT NULL,
    FirstName varchar(25) NOT NULL,
    PLZ       numeric(4)  REFERENCES Ort (Zip)
)
```

Exercise 2.1: Course registration



Think: What could be the PK for Registration

Course

CourseID	CourseName
GEO874	DB
GEO800	GIS 1
GEO801	GIS II

Primary Key CourseID

Registration

CourseNr	StuID	RegistratonDate
GEO874	13-888-888	2011-11-10 10:55

Foreign Key StuID

Foreign Key CourseNr

Student

StudentID	FirstName	LastName	PLZ	Gender
13-888-888	Stofer	Christian	8001	M
13-888-889	Ginzler	Silvia	8057	NULL

Primary Key StudentID

Think: What could be the PK for Registration

Course

CourseID	CourseName
GEO874	DB
GEO800	GIS 1
GEO801	GIS II

Primary Key CourseID

Registration

CourseNr	StuID	RegistratonDate
GEO874	13-888-888	2011-11-10 10:55

Foreign Key StuID

Foreign Key CourseNr

Primary Key (CourseNr, StuID)

Student

StudentID	FirstName	LastName	PLZ	Gender
13-888-888	Stofer	Christian	8001	M
13-888-889	Ginzler	Silvia	8057	NULL

Primary Key StudentID

Alteration of Tables

ALTER TABLE – alters the definition of a table

- Definition or removal of a column (Attribute)
- Alteration of a column definition (e.g., data type)
- Definition or removal of constraints

```
ALTER TABLE user01.Ort DROP COLUMN PLZ CASCADE;
```

Needed if referential constraints have been defined

Removes the column PLZ from the table Ort

```
ALTER TABLE user01.Person ADD COLUMN Street varchar(30);
```

- Adds the column Street in Table Person
- After creation, all the values in Street are NULL

- Changes in data type definition
  - PostgreSQL will attempt to convert the column's existing values (if any) to the new type.
    - varchar(30) to varchar(32): possible
    - varchar(30) to varchar(28): will fail or produce surprising results

➲ https://www.postgresql.org/docs/12/ddl-alter.html

Deletion of Tables

DROP TABLE – Discards a table (contents/data and schema)

```
DROP TABLE user01.Person
```

Before you do, beware:

- All associated data, indexes, triggers and constraints will also be discarded;
- To drop a table that is referenced by a view or a foreign-key constraint of another table, CASCADE must be specified
  - CASCADE will remove a dependent view entirely,
  - but in the foreign-key case it will only remove the foreign-key constraint, not the other table entirely.
- To only discard DATA, use DELETE * FROM... (see later)

Options for dropping tables

Person

PersNr	LastName	FirstName	PLZ
1	Von Gunten	Reto	3000
2	Stofer	Christian	6330
3	Stofer	Silvia	6330
4	Ginzler	Christian	9000
5	Burghardt	Dirk	8001

Ort

Zip	Location
6330	Cham
9000	St. Gallen
8002	Zürich
6300	Zug
8400	Winterthur
8003	Zürich
3000	Bern
8001	Zürich
8051	Zürich

- RESTRICT – disables the deletion of the table if it has dependent objects (default setting if not specified)

```
DROP TABLE user01.Ort RESTRICT;
```

- Nothing happens, Ort not dropped

- CASCADE – deletes automatically all linked dependent views, attributes and constraints (but not the tables themselves).

```
DROP TABLE user01.Ort CASCADE;
```

- Ort is dropped, incl. FK constraint in Person

Exercise 2.2

Assume that the University allows Students to pay with their UZH Card for lunches in the mensa. They charge it with a certain amount, and after each use the set amount will be deducted. Set up the table Card_Transaction with the attributes and their parameters as defined below. Select appropriate data types and discuss with neighbour (3 min.)

- Attributes:
  - student name,
  - gender,
  - card number,
  - starting value,
  - Balance (value left)
  - Charge date
  - Personal Identification Number (PIN).
- Assumptions:
  - The attribute “student name” has max 25 characters.
  - The attribute “gender” might not be specified.
  - The attribute “value left” and “starting value” must not be smaller than 0.
  - The attribute “card number” can have up to 15 characters.
  - The attribute “charge date” consists of a date and a time
  - The attribute “PIN” might not be specified, and if specified, it is always 12 digits long.

Exercise 2.2

- Assumptions:
  - The attribute “student name” has max 25 characters.
  - The attribute “gender” might not be specified.
  - The attribute “value left” and “starting value” must not be smaller than 0.
  - The attribute “card number” can have up to 15 characters.
  - The attribute “charge date” consists of a date and a time
  - The attribute “PIN” might not be specified, and if specified, it is maximum12 digits long.

```
CREATE TABLE Card_Transaction (
  Student_Name     varchar(  )
  Gender           char(1),
  Card_Number      varchar(  )
  Starting_Value   numeric(6,2)                                    ,
  Value_Left       numeric(6,2)
  Charge_Date      timestamp
  Pin_Number       numeric(  )
);
```

Pen & Pencil time: try to fill the blanks. Some blanks can be null.

3 mins

Solution Exercise 2.2

```
CREATE TABLE Card_Transaction (
  Student_Name     varchar(25) NOT NULL,
  Gender           char(1),
  Card_Number      varchar(15) NOT NULL,
  Starting_Value   numeric(6,2) NOT NULL CHECK (Starting_Value>=0),
  Value_Left       numeric(6,2) NOT NULL CHECK (Value_Left>=0),
  Charge_Date      timestamp NOT NULL,
  Pin_Number       numeric(12)
);
```

SQL – the language for relational databases

1. SQL-Intro
2. Data types
3. SQL concepts: database, schema, table
4. Data Definition Language (DDL)
5. **Data Manipulation Language (DML)**
6. Data Query Language (DQL)
7. Definition of Views

Data Manipulation Language (DML)

Modification of the data in a table

- INSERT
- DELETE
- UPDATE

Modification of data in a table

INSERT – load data

```
INSERT INTO user01.Person VALUES(13, 'Niederberger', 'Thomas', 8001);
```

Insert values for all columns

```
INSERT INTO user01.Person(PersNr, LastName, FirstName)
VALUES(13, 'Niederberger', 'Thomas');
```

A subset of columns (needs to specifiy the column names)

A DBMS manages integrity constraints – e.g., checks whether a person with PK 13 exists (and therefore, this 2nd SQL code will not work)

The following fails to insert into Person as the PK PersNr is not specified (against NOT NULL constraint)

```
INSERT INTO user01.Person(LastName, FirstName)
VALUES('Niederberger', 'Thomas');
```

Modification of data in a table

UPDATE – modifies rows

```
UPDATE      user01.Person
SET         PLZ=8051, street=´Winterthurerstrasse´
WHERE       LastName=´Niederberger´;
```

Changes PLZ and streetname for people with LastName „Niederberger“

DELETE – deletes rows in a table

```
DELETE FROM   user01.Person
WHERE  LastName=´Niederberger´;
```

The WHERE clause specifies which row(s) will be deleted

SQL – the language for relational databases

1. SQL-Intro
2. Data types
3. SQL concepts: database, schema, table
4. Data Definition Language (DDL)
5. Data Manipulation Language (DML)
6. **Data Query Language (DQL)**
7. Definition of Views

Querying data: SELECT - statement

Basic SQL statement for data querying: SELECT

```
SELECT <Column1>, <Column2>, …
FROM   <TableA>
WHERE  <Column1> = …;
```

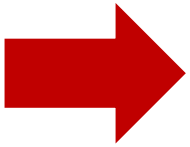
SFW-Block

SELECT Example

Example: Who lives in PLZ „6330“?

Person

PersNr	LastName	FirstName	PLZ
1	Von Gunten	Reto	3000
2	Stofer	Christian	6330
3	Stofer	Silvia	6330
4	Ginzler	Christian	9000
5	Burghardt	Dirk	8001
6	Suter	Christoph	8051
7	Meile	Rolf	8400
8	Weibel	Robert	8003



```
SELECT LastName, FirstName
FROM Person
WHERE PLZ = 6330
```

LastName	LastName
Stofer	Christian
Stofer	Silvia

set-oriented

Check one by one.
What if there are several millions records?
Will take a long time!!

Indexes can help!

SFW-Block

```
SELECT <Column1>, <Column2>, …
FROM   <TableA>
WHERE  <Column1> = … ;
```

SELECT

- Specifies the projection list of the resulting schema (list of attributes)
- Integrates also arithmetical operators and aggregate functions

FROM

- specifies tables or views on which the SELECT is applied
- Applies aliases, if specified
- If multiple tables are specified, the result is a cartesian product

WHERE

- Specifies selection conditions
- Specifies linkage conditions to compute JOINs of the mentioned cartesian products
- You can nest queries in where clauses – you can have an entire SFW block inside a WHERE clause

SFW-Block

```
SELECT LastName, FirstName, CityName
FROM   Person, City;
```

👉 Every row in `Person` gets combined with every row in `City`.
If you had 1,000 people and 500 cities, you'd suddenly have **500,000 rows**.
Cartesian product.

```
SELECT LastName, FirstName, CityName
FROM   Person, City
WHERE  Person.PLZ = City.Zip;
```

👉 Now it only returns the rows where the `PLZ` in the person table matches the `Zip` in the city table. This is what we usually want: each person is linked to their correct city.

Qualified names and aliases

- Tables can be accessed using qualified names (schema.table). This is necessary when accessing tables from a different schema (e.g., “user02”):

```
SELECT LastName, FirstName
FROM user01.Person
WHERE PLZ = 8003
```

- Qualified names can be also used to address columns:

```
SELECT Person.LastName, Person.FirstName
FROM user01.Person
WHERE Person.PLZ= 8003
```

```
SELECT p.LastName, p.FirstName
FROM user01.Person p
WHERE p.PLZ = 8003
```

Alias: Variable p for user01.Person

Relational Algebra: PROJECT (operator)

- PROJECT isolates specified columns from a table

$$\text{PROJECT}_{\text{LastName, FirstName}}\ (\text{Person})$$

Relational Algebra: PROJECT

```
SELECT LastName, FirstName
FROM Person;
```

SQL: SELECT statement!

LastName	FirstName
Von Gunten	Reto
Stofer	Christian
Stofer	Silvia
Ginzler	Christian
Burghardt	Dirk

5 rows returned

Relational Algebra: SELECTION

- SELECTION *filters* specified rows in the table

$$\text{SELECT}_{\text{LastName='Ginzler'}}\ (\text{Person})$$

Relational Algebra: SELECT

```
SELECT *
FROM Person
WHERE LastName='Ginzler';
```

SQL: WHERE clause

PersNr	LastName	FirstName	PLZ
1	Von Gunten	Reto	3000
2	Stofer	Christian	6330
3	Stofer	Silvia	6330
4	Ginzler	Christian	9000
5	Burghardt	Dirk	8001

1 row returned

Cartesian product (projection on multiple tables)

- If more than one relation (table) is involved, the Cartesian product will be built

Person

PersNr	LastName	FirstName	PLZ
1	Von Gunten	Reto	3000
2	Stofer	Christian	6330
3	Stofer	Silvia	6330

Ort

Code	Location
6330	Cham
9000	St. Gallen
8002	Zürich

```
SELECT *
FROM user01.Person,user01.Ort
```

PersNr	LastName	FirstName	PLZ	Code	Location
1	Von Gunten	Reto	3000	6330	Cham
1	Von Gunten	Reto	3000	9000	St. Gallen
1	Von Gunten	Reto	3000	8002	Zürich
2	Stofer	Christian	6330	6330	Cham
2	Stofer	Christian	6330	9000	St. Gallen
2	Stofer	Christian	6330	8002	Zürich
3	Stofer	Silvia	6330	6330	Cham
3	Stofer	Silvia	6330	9000	St. Gallen
3	Stofer	Silvia	6330	8002	Zürich

3 X 3 = 9 rows returned

JOIN = Cartesian product + SELECT-filtering

- Same as: building a Cartesian product and then applying the filter

Person

PersNr	LastName	FirstName	PLZ
1	Von Gunten	Reto	3000
2	Stofer	Christian	6330
3	Stofer	Silvia	6330

Ort

Code	Location
6330	Cham
9000	St. Gallen
8002	Zürich

```
SELECT *
FROM user01.Person,user01.Ort
WHERE Person.PLZ = Ort.Code
```

PersNr	LastName	FirstName	PLZ	Code	Location
1	Von Gunten	Reto	3000	6330	Cham
1	Von Gunten	Reto	3000	9000	St. Gallen
1	Von Gunten	Reto	3000	8002	Zürich
2	Stofer	Christian	6330	6330	Cham
2	Stofer	Christian	6330	9000	St. Gallen
2	Stofer	Christian	6330	8002	Zürich
3	Stofer	Silvia	6330	6330	Cham
3	Stofer	Silvia	6330	9000	St. Gallen
3	Stofer	Silvia	6330	8002	Zürich

2 rows returned

Different types of JOIN

- Join type
  - INNER JOIN (a.k.a. JOIN)
    - The previous example is an INNER JOIN.
  - LEFT OUTER JOIN (a.k.a. LEFT JOIN)
  - RIGHT OUTER JOIN (a.k.a. RIGHT JOIN)
  - FULL OUTER JOIN (a.k.a. FULL JOIN)

- T1 join_type T2 ON [join_condition]
  - Join condition normally compares two columns, which must be of the same data type. (Number = Number, String = String)
  - Most often „Foreign Key“ = „Primary Key“

https://www.postgresql.org/docs/12/queries-table-expressions.html

INNER JOIN

- The INNER JOIN keyword selects all rows from both tables as long as there is a match between the columns.

```
SELECT *
FROM user01.Customers, user01.Orders
where Customers.CID=Orders.CID
```

```
SELECT *
FROM user01.Customers INNER JOIN user01.Orders
ON Customers.CID=Orders.CID
```

```
SELECT *
FROM user01.Customers JOIN user01.Orders
ON Customers.CID=Orders.CID
```

```
SELECT *
FROM user01.Customers NATURAL INNER JOIN user01.Orders
```

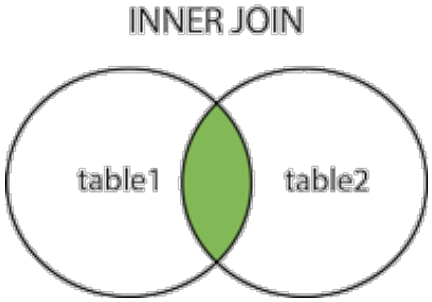


Table: Customers

CID	Name	Country
1	Alfreds Futter	Germany
2	Ana Trujillo	Mexico
3	Antonio Moreno	Mexico

Table: Orders

OID	CID	OrderDate
10308	2	9/18/1996
10309	37	9/19/1996
10310	77	9/20/1996

results

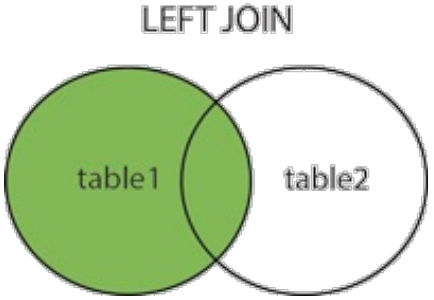
Customers.CID	Customers.Name	Customers.Country	Orders.OID	Orders.CID	Orders.OrderDate
2	Ana Trujillo	Mexico	10308	2	9/18/1996

What happens if no matching values are found?

-- will return 0 row

LEFT OUTER JOIN

- The LEFT JOIN (LEFT OUTER JOIN) keyword returns all rows from the left table (table1), with the matching rows in the right table (table2). The result is NULL in the right side when there is no match.




```
SELECT *
FROM user01.Customers LEFT JOIN user01.Orders
ON Customers.CID=Orders.CID
```

```
SELECT *
FROM user01.Customers NATURAL LEFT JOIN user01.Orders
```

Table: Customers

CID	Name	Country
1	Alfreds Futter	Germany
2	Ana Trujillo	Mexico
3	Antonio Moreno	Mexico

Table: Orders

OID	CID	OrderDate
10308	2	9/18/1996
10309	37	9/19/1996
10310	77	9/20/1996

results

Customers.CID	Customers.Name	Customers.Country	Orders.OID	Orders.CID	Orders.OrderDate
1	Alfreds Futter	Germany	NULL	NULL	NULL
2	Ana Trujillo	Mexico	10308	2	9/18/1996
3	Antonio Moreno	Mexico	NULL	NULL	NULL

RIGHT OUTER JOIN

- The RIGHT JOIN (RIGHT OUTER JOIN) keyword returns all rows from the right table (table2), with the matching rows in the left table (table1). The result is NULL in the left side when there is no match.

```
SELECT *
FROM Customers RIGHT JOIN Orders
ON Customers.CID=Orders.CID
```

```
SELECT *
FROM Customers NATURAL RIGHT JOIN Orders
```

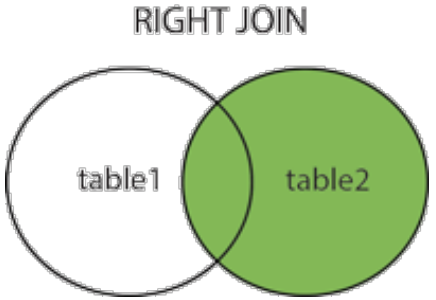


Table: Customers

CID	Name	Country
1	Alfreds Futter	Germany
2	Ana Trujillo	Mexico
3	Antonio Moreno	Mexico

Table: Orders

OID	CID	OrderDate
10308	2	9/18/1996
10309	37	9/19/1996
10310	77	9/20/1996

results

Customers.CID	Customers.Name	Customers.Country	Orders.OID	Orders.CID	Orders.OrderDate
2	Ana Trujillo	Mexico	10308	2	9/18/1996
NULL	NULL	NULL	10309	37	9/19/1996
NULL	NULL	NULL	10310	77	9/20/1996

FULL OUTER JOIN

- The FULL OUTER JOIN (FULL JOIN) keyword returns all rows from the left table (table1) and from the right table (table2).
- The FULL OUTER JOIN keyword combines the result of both LEFT and RIGHT joins.

```
SELECT *
FROM user01.Customers FULL JOIN user01.Orders
ON Customers.CID=Orders.CID
```

```
SELECT *
FROM user01.Customers NATURAL FULL JOIN user01.Orders
```

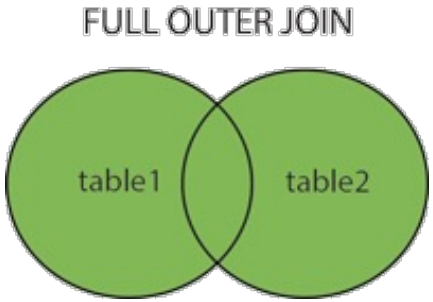


Table: Customers

CID	Name	Country
1	Alfreds Futter	Germany
2	Ana Trujillo	Mexico
3	Antonio Moreno	Mexico

Table: Orders

OID	CID	OrderDate
10308	2	9/18/1996
10309	37	9/19/1996
10310	77	9/20/1996

results

Customers.CID	Customers.Name	Customers.Country	Orders.OID	Orders.CID	Orders.OrderDate
1	Alfreds Futter	Germany	NULL	NULL	NULL
2	Ana Trujillo	Mexico	10308	2	9/18/1996
3	Antonio Moreno	Mexico	NULL	NULL	NULL
NULL	NULL	NULL	10309	37	9/19/1996
NULL	NULL	NULL	10310	77	9/20/1996

The FULL OUTER JOIN keyword returns all the rows from the left table (Customers), and all the rows from the right table (Orders). If there are rows in "Customers" that do not have matches in "Orders", or if there are rows in "Orders" that do not have matches in "Customers", those rows will be listed as well.

Exercise 2.3

Table1	
A	**BL**
1	2
2	3

Table2	
BR	**C**
3	4
4	5

Join condition:
Table1.BL=Table2.BR

INNER JOIN

?

LEFT OUTER JOIN

?

FULL OUTER JOIN

?

RIGHT OUTER JOIN

?

Self Join

- Requires different aliases for the same table
- E.g.,: Person a, Person b

Person

PersNr	LastName	FirstName	PLZ	Assistant_of
1	Von Gunten	Reto	3000	(null)
2	Burghardt	Dirk	8001	8
3	Suter	Christoph	8051	(null)
4	Meile	Rolf	8400	(null)
8	Weibel	Robert	8003	(null)

```
SELECT a.FirstName FN_Asst,
a.LastName LN_Asst, a.Assistant_of,
b.FirstName FN_Prof, b.LastName LN_Prof
FROM user01.Person a LEFT JOIN
     user01.Person b
ON a.Assistant_of = b.PersNr
```

FN_Asst	LN_Asst	Assistant_of	FN_Prof	LN_Prof
Reto	Von Gunten	(null)	(null)	(null)
Dirk	Burghardt	8	Robert	Weibel
Christoph	Suter	(null)	(null)	(null)
Rolf	Meile	(null)	(null)	(null)
Robert	Weibel	(null)	(null)	(null)

SQL – the language for relational databases

1. SQL-Intro
2. Data types
3. SQL concepts: database, schema, table
4. Data Definition Language (DDL)
5. Data Manipulation Language (DML)
6. Data Query Language (DQL)
7. **Definition of Views**

Views (virtual tables) in SQL

- A view is a virtual table based on the result-set of an SQL statement (e.g., a “SELECT statement”).

- A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.
- You can use a view like a real table in SQL queries.

View

- Create/delete a view

```
CREATE VIEW user01.test1 AS
    SELECT *
    FROM        user01.student
    WHERE       Course = 'Math'
```

```
DROP VIEW user01.test1
```

Table: student

Name	Course	Grade
A	Math	80
A	English	90
A	Physics	70
B	Math	75
B	English	75
B	Physics	75
C	Math	60
C	English	80
C	Physics	50

- Use a view

```
SELECT * FROM user01.test1
```

The view is **not physically materialized/stored**. Instead, the query is run every time the view is referenced in a query.

A view always shows **up-to-date data**! The database engine recreates the data, using the view's SQL statement, every time a user queries a view.

Updatable Views

- Simple views are automatically updatable: support INSERT, UPDATE and DELETE statements
- A view is automatically updatable if it satisfies all of the following conditions:
  - The view must have exactly one entry in its FROM list, which must be a table or another updatable view.
  - The view definition must not contain WITH, DISTINCT, GROUP BY, HAVING, LIMIT, or OFFSET clauses at the top level.
  - The view definition must not contain set operations (UNION, INTERSECT or EXCEPT) at the top level.
  - All columns in the view's select list must be simple references to columns of the underlying table. They cannot be expressions or functions.
  - No column of the underlying table can appear more than once in the view's select list.
- **If the view is automatically updatable, the system will convert any INSERT, UPDATE or DELETE statement on the view back into the corresponding statement on the underlying tables.**

Details

Why views?

- In some cases, it is not desirable for all users to see the entire logical model (that is, all the actual tables stored in the database).

- Consider a user who needs to know an instructor's name and department, but not the salary. This user should see a table described, in SQL, by

```
CREATE VIEW user01.test1 AS
SELECT ID, name, dept_name
FROM user01.instructor
```

- A view provides a mechanism to hide certain data from the view of certain users.

Summary 2.1

SQL-Element	SQL-Statement	
Data definition	CREATE TABLE	**DDL**
	ALTER TABLE	
	DROP TABLE	
	CREATE INDEX	
	DROP INDEX	
Data manipulation	INSERT INTO	**DML**
	UPDATE	
	DELETE	
Querying data	SELECT, JOIN	**DQL**

Summary 2.2

- SQL is the standard language for the definition, querying and manipulation of data in relational databases.
- SQL contains the following sub-languages: Data Definition Language (DDL), Data Manipulation Language (DML), Data Query Language (DQL), Data Control Language (DCL), and Transaction Control Language (TCL).
- Join-Operations allow the combination of tuples from multiple relations (Tables) into a single tuple.

Exercises: SQL Tutorial at W3Schools

http://www.w3schools.com/sql/

Later, in lab

- 10:15-12:00

- Y25-J-09, Y25-J-10

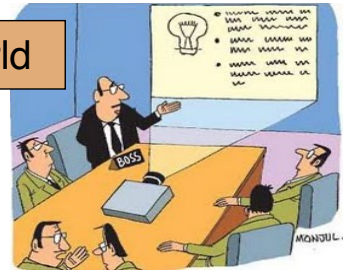
- **SQL demo**
- **SQL Exercise**

Next weeks: More SQL & DB Design

Lecture 3 — More SQL & Requirements Analysis

Interview of users, study of documentation

Real World



Entities in the real world: Department head Müller, Employee Muster, Project 'Budget 2014'

Database Requirements

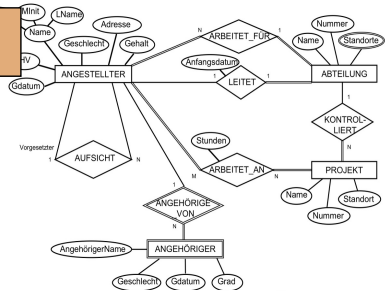
1. Die Firma ist in Abteilungen organisiert. Jede Abteilung hat eine eindeutige Bezeichnung, eine eindeutige Nummer und einen bestimmten Angestellten, der sie leitet. Wir registrieren das Anfangsdatum, ab dem dieser Angestellte die Abteilung übernommen hat. Eine Abteilung verfügt über mehrere Standorte.
2. Eine Abteilung kontrolliert eine Reihe von Projekten, die jeweils einen eindeutigen Namen und eine eindeutige Nummer und einen einzigen Standort haben.
3. Jeder Angestellte wird mit Namen, AHV-Nr., Adresse, Gehalt, Geschlecht und Geburtsdatum gespeichert. Ein Angestellter wird einer Abteilung zugewiesen, kann aber an mehreren Projekten arbeiten, die nicht unbedingt alle von der gleichen Abteilung kontrolliert werden. Wir verfolgen die Stundenzahl pro Woche, die ein Angestellter an jedem Projekt arbeitet, und den unmittelbaren Vorgesetzten jedes Angestellten.
4. Zu Versicherungszwecken sollen die Familienangehörigen jedes Mitarbeiters mit Vorname, Geschlecht, Geburtsdatum und Verwandtschaftsgrad zum jeweiligen Angestellten erfasst werden.

Written concise description of requirements (data reqs and functional [operations] reqs)

Lecture 3/4 — Conceptual DB design

Text analysis
Entities
Relationships

ER-Model



Entity types with attributes and relationships

DBMS-independent
DBMS-specific

Lecture 5 — Logical DB design (data model mapping)

- translation of model to implementation
- Validation through normalisation

DB-Schema (e.g. relational data model)

ANGESTELLTER

VNAME	INITAL	NNAME	AHV	GDATUM
ADRESSE	GESCHLECHT	GEHALT	SUPERAHV	ANR

ABTEILUNG

ANAME	ABTNUMMMER	MGRAHV	MGR_ANFANGSDATUM

ABT_STANDORTE

ABTNUMMMER	ASTANDORT

ARBEITET_AN

EAHV	PNR	STUNDEN

PROJEKT

PNAME	PNUMMER	PSTANDORT	ABTNR

ANGEHÖRIGER

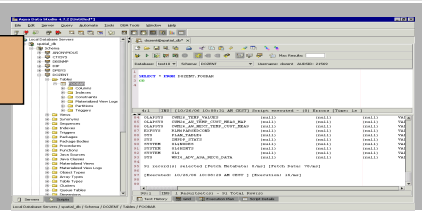
EAHV	ANGEHÖRIGER_NAME	GESCHLECHT	GDATUM	GRAD

Set of relations with attributes (incl. data types + value domains)

Lecture 5 — Physical DB design

Description of database implementation (HW, indexes, …)

Database



Implemented DB design, Tables (ordered) with rows and columns

➲ Elmasri & Navathe (2014)