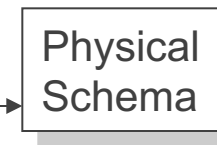
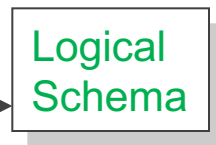
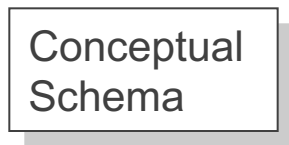
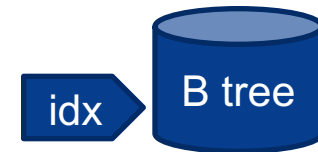
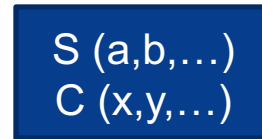
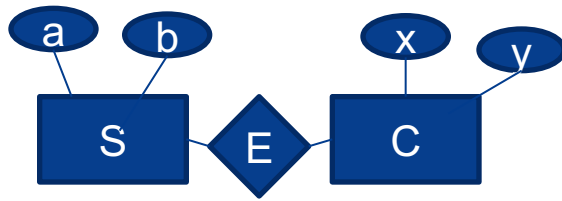


Relational Model to the DBMS (via SQL DDL)

Data can be represented at different levels of abstraction



ER:

- Entities,
- Relationships,
- Attributes

Relations:

- tuples,
- attributes
- domain

File organisation:

- File types
- Index structures

≈ Tables/columns/rows

specifically tied to the data model of a database you chose (for our course, the choice is relational database)

specifically tied to the implementation
Methods provided by the database you chose

Not specifically tied to a database

What is RDBMS?

A *relational database management system* (RDBMS) is

- software designed to support large-scale data-intensive applications
- **allowing high-level description of data (tables, constraints)**
- **with high-level access to the data (relational model, SQL)**
- providing efficient storage and retrieval (disk/memory management)
- supporting multiple simultaneous users (privilege, protection)
- doing multiple simultaneous operations (transactions, concurrency)
- maintaining reliable access to the stored data (backup, recovery)

Note: databases provide *persistent* storage of information

Describing Data – SQL language

Relational Database Management Systems (RDBMS) implement \cong the relational model.

SQL provides the high-level access to describing and accessing data

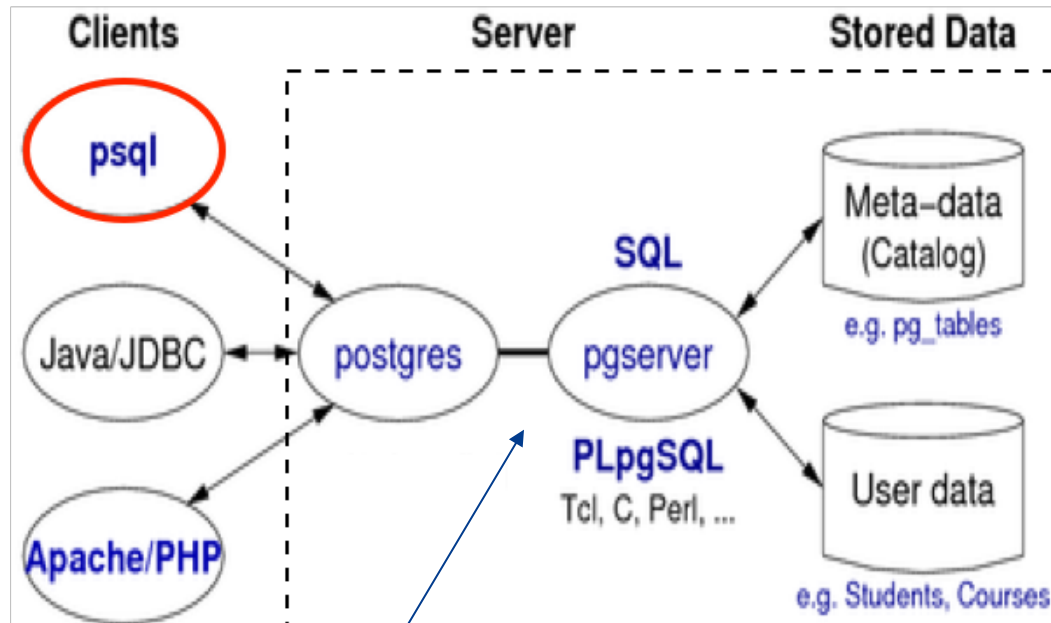
In RDBMS, Using SQL, one can define:

- domains, attributes, tuples, tables
- constraints (domain, key, referential)

Variations from the relational model:

- no strict requirement for tables to have keys
- bag semantics, rather than set semantics
 - set: no order, without duplicates vs. bag: no order, with duplicates
 - *if no key is defined in an SQL table, duplicate tuples can exist*
- no standard support for general (multi-table) constraints
 - *use advanced concepts like triggers instead*

PostgreSQL client-server architecture



- Originally "Structured Query Language", today a proper name
- A language with several functionalities, comprises of :
 - DDL (Data Definition Language)
 - DML (Data Manipulation Language)
- There exists several standards, and companies have added proprietary extensions

Managing Databases

Shell commands:

- **createdb** *dbname*
- **dropdb** *dbname*

(If no *dbname* supplied, assumes a database called *YOU*)

SQL DDL (Data Definition Language) statements (inside *dbname*)

- **CREATE TABLE** *table* (*Attributes+Constraints*)
- **ALTER TABLE** *table* *TableSchemaChanges*
- **DROP TABLE** *table(s)* [**CASCADE**]
- **COPY** *table* (*AttributeName*) **FROM STDIN**
- **INSERT INTO** *table* (*attrs*) **VALUES** *tuple(s)*
- **DELETE FROM** *table* **WHERE** *condition*
- **UPDATE** *table* **SET** *AttrValueChanges* **WHERE** *condition*

Data Definitions in SQL

CREATE TABLE

- Defines a relation schema (with attributes and integrity constraints)
- Creates an empty instance of the schema

```
create table Employee (  
    EmpNo          char(6) primary key,  
    FirstName      varchar(20) not null,  
    LastName       varchar(20) not null,  
    Dept           varchar(15),  
    Salary         numeric(9) default 0,  
    City           varchar(15),  
    foreign key(Dept) references Department(DeptName),  
    unique (LastName,FirstName)  
)
```

Domains

Elementary domains or types (predefined)

- Character: single characters or strings → both of fixed and variable length
- Bitstrings: string elements are 0 and 1
- Numbers: integers and reals
- Dates, timestamps, time intervals
- Introduced in SQL:1999
 - Boolean
 - BLOB, CLOB (binary/character large object): for large images or texts
- In some systems, enumeration types can be defined

User defined domains (reusable)

Domain Definitions

User defined domains

create domain

- 1) defines a (simple) domain with integrity constraints and defaults, which can be reused in table definitions.
- 2) data types based on a constrained version of an existing data type;

Example:

```
create domain EmployeeAge
as smallint default null
check ( value >=18 and value <= 67 )
```

```
create table Employee (
    EmpNo      char(6) primary key,
    Age        EmployeeAge,
    Name       varchar(20),
    ...)
```


Constraints on a Relation

not null (on single attributes)

unique: allows one to define a (candidate) key:

- on *single attribute*
- on *several attributes* (i.e., one or more): `unique (Attr1, ..., Attrn)`

primary key: definition of the primary key (implies not null)

```
create table Employee (  
    EmpNo           char(6) primary key,  
    FirstName      varchar(20) not null,  
    LastName       varchar(20) not null,  
    Dept           varchar(15),  
    Salary         numeric(9) default 0,  
    City           varchar(15),  
    foreign key (Dept) references Department(DeptName),  
    unique (LastName,FirstName)  
)
```

Primary key – two ways to define it ...

```
create table Employee (  
    EmpNo          char(6) primary key,  
    FirstName      varchar(20) not null,  
    LastName       varchar(20) not null,  
    Dept           varchar(15),  
    Salary         numeric(9) default 0,  
    City           varchar(15),  
    foreign key (Dept) references Department(DeptName),  
    unique (LastName,FirstName)  
)
```

```
create table Employee (  
    EmpNo          char(6),  
    FirstName      varchar(20) not null,  
    LastName       varchar(20) not null,  
    Dept           varchar(15),  
    Salary         numeric(9) default 0,  
    City           varchar(15),  
    primary key (EmpNo),  
    foreign key (Dept) references Department(DeptName),  
    unique (LastName,FirstName)
```

Candidate keys: mind the step!

```
create table Employee ( ...  
    FirstName  varchar(20) not null,  
    LastName   varchar(20) not null,  
    unique (LastName,FirstName)  
)
```

is **different** from:

```
create table Employee ( ...  
    FirstName  varchar(20) not null unique,  
    LastName   varchar(20) not null unique  
)
```

Constraints Between Relations

references and **foreign key** allow one to define referential integrity constraints.

Syntax:

- for single attributes:
 references after the specification of the domain
- for several attributes:
 foreign key(Attr₁,...,Attr_n) references ...

The referenced attributes in the target table must form a key (primary key or unique). If they are missing, the foreign key refers to the primary key of the target table.

Semantics: every combination (not involving NULL) of attribute values in the source table must appear in the target table.

It is possible to add policies that specify how to react to constraint violations (which are caused by changes of the target table).

Constraints Between Relations

```
create table Student(  
  StudNo  character(10) primary key,  
  Name    character(20),  
  Hons    character(3),  
  Tutor   character(20) references Staff(Lecturer),  
  Year    smallint)
```

```
create table Staff(  
  Lecturer  character(20) primary key,  
  RoomNo    character(4),  
  Appraiser character(20),  
  foreign key (Appraiser) references  
    Staff(Lecturer)  
    on delete set null  
    on update cascade)
```

Policies

Determine the effect of delete and update statements

Syntax

- on delete Action

where Action can be

cascade	(propagate the deletion)
restrict	(do nothing if the row is referenced)
no action	(as restrict, but return an error)
set default	
set null	

The same actions exist for updates (on update)

Schema Updates

`alter domain`: allows one to modify a domain definition

`alter table`: allows one to modify a table

- add or drop attributes
- add or drop constraints

`drop domain`: eliminates a domain

`drop table`: eliminates a table