PostgreSQL

How To Migrate From Other Database Systems To PostgreSQL

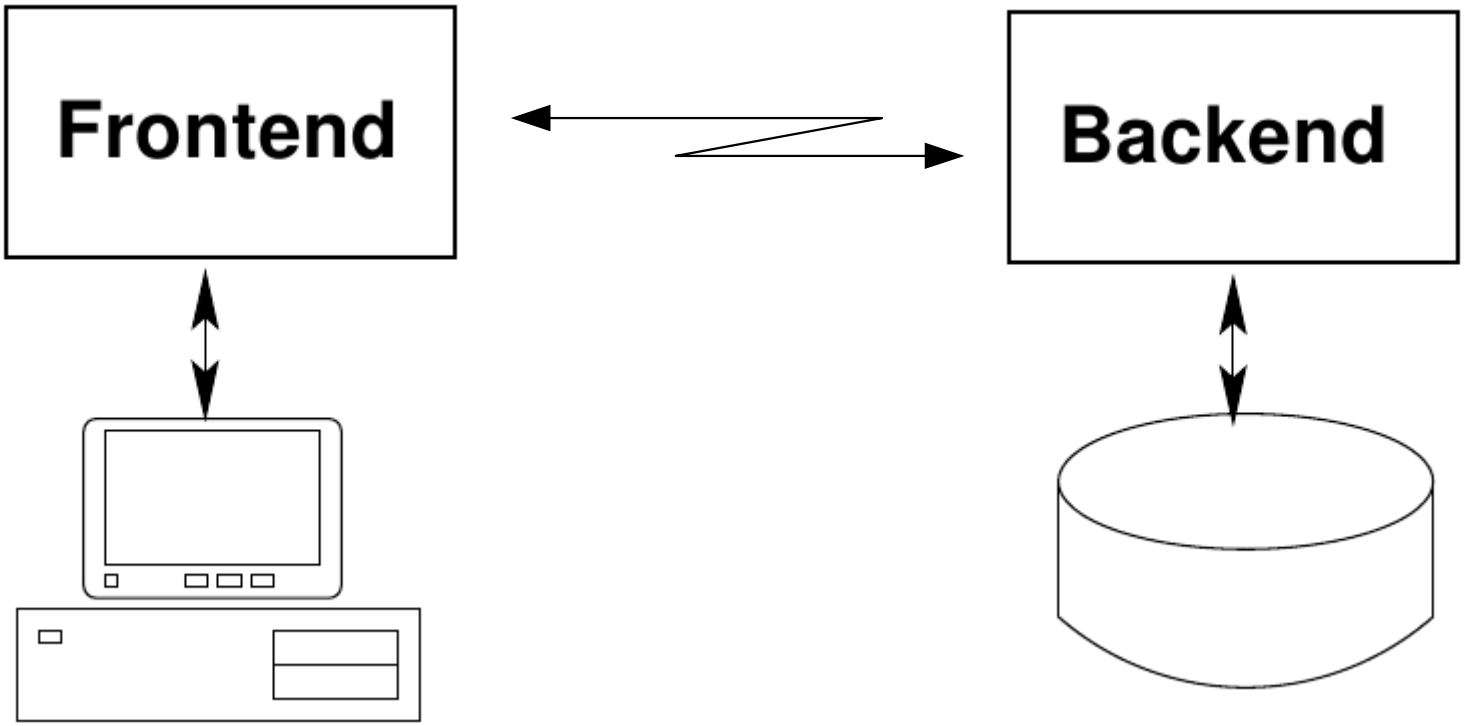
Making use of the features of PostgreSQL – thus making your application programming easier

Holger@Jakobs.com
FrOSCon 2011

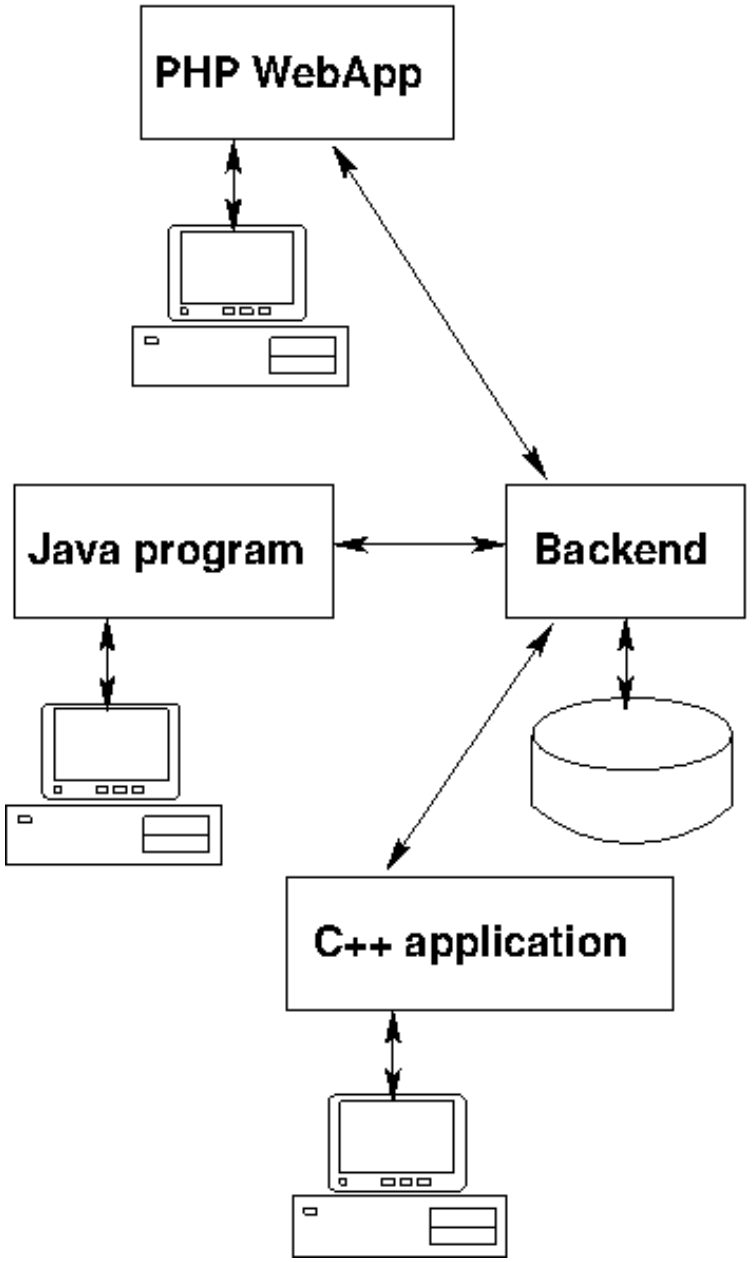
PostgreSQL at a glance

- Other database systems are more popular, but not more advanced.
- NoSQL databases are no alternative at all if you need to save structured data – as opposed to documents.
- PostgreSQL aims to be as close to the SQL standard as possible.
- Let's see how PostgreSQL can help you to make application programming easier and more failsafe.

Database and Application



Database and Application



Frontend may be:

- traditional application directly accessing database server (C, C++, Java, ...)
- web application (Java, PHP, ...)
- or maybe both?
- or even more?

Database and Application

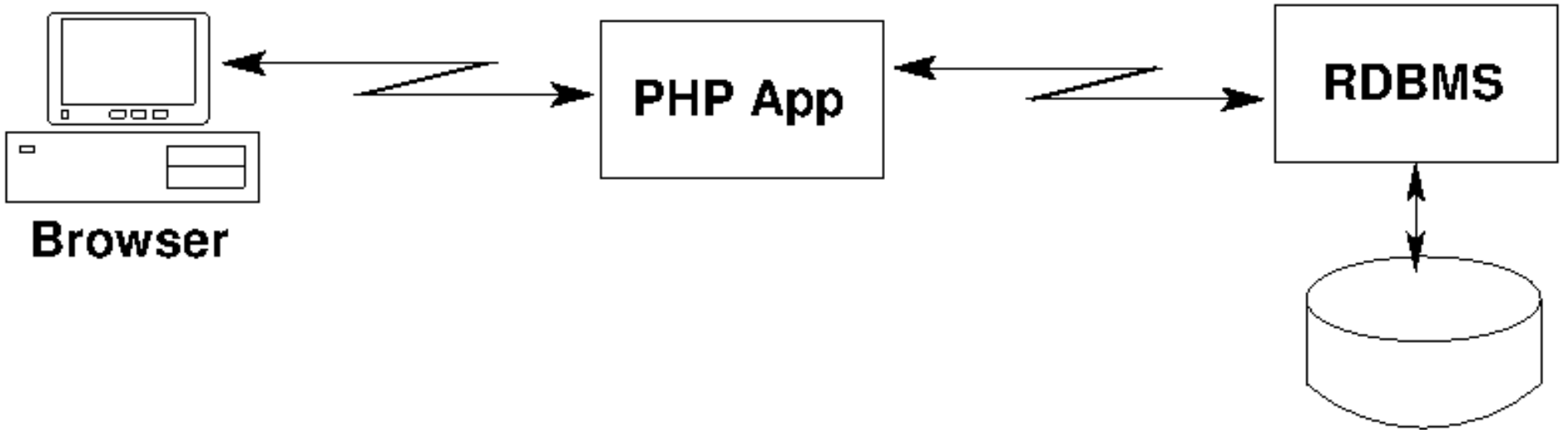
Conclusion:

If you have – or might develop over time – several frontends to your database, doing integrity checks in your application program becomes difficult.

The checks ...

- have to programmed again and again.
- might not all do exactly the same thing.
- will bloat your applications – especially problematic on mobile platforms.
- won't help while using an interactive tool.

Typical PHP Application

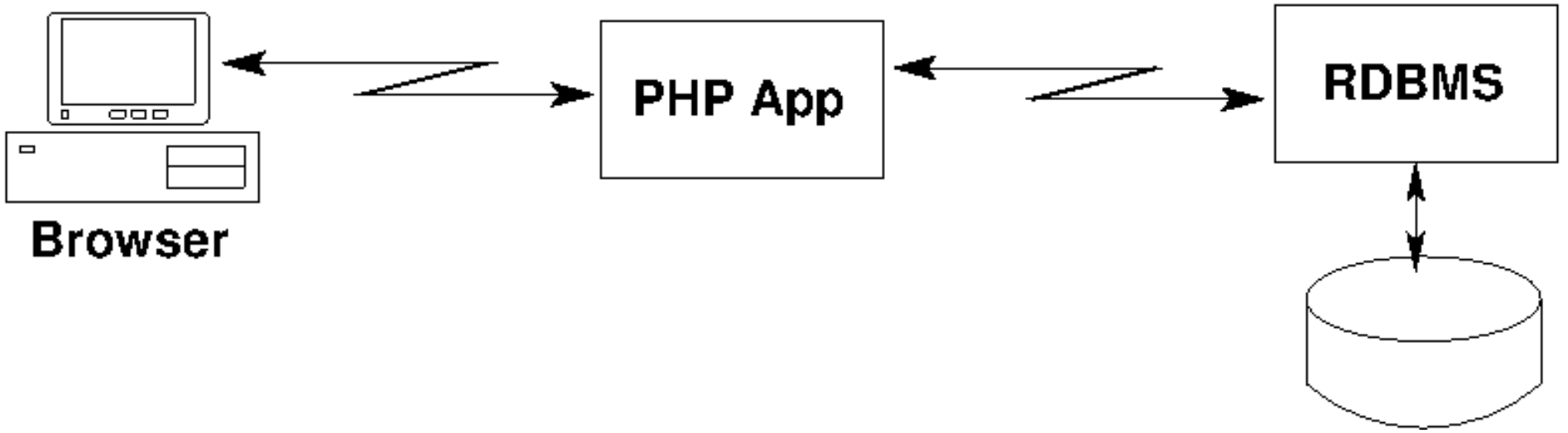


user interface, minimal plausibility checks

mixture of application logic, plausibility and integrity checks

used as pure data storage

Typical PHP Application



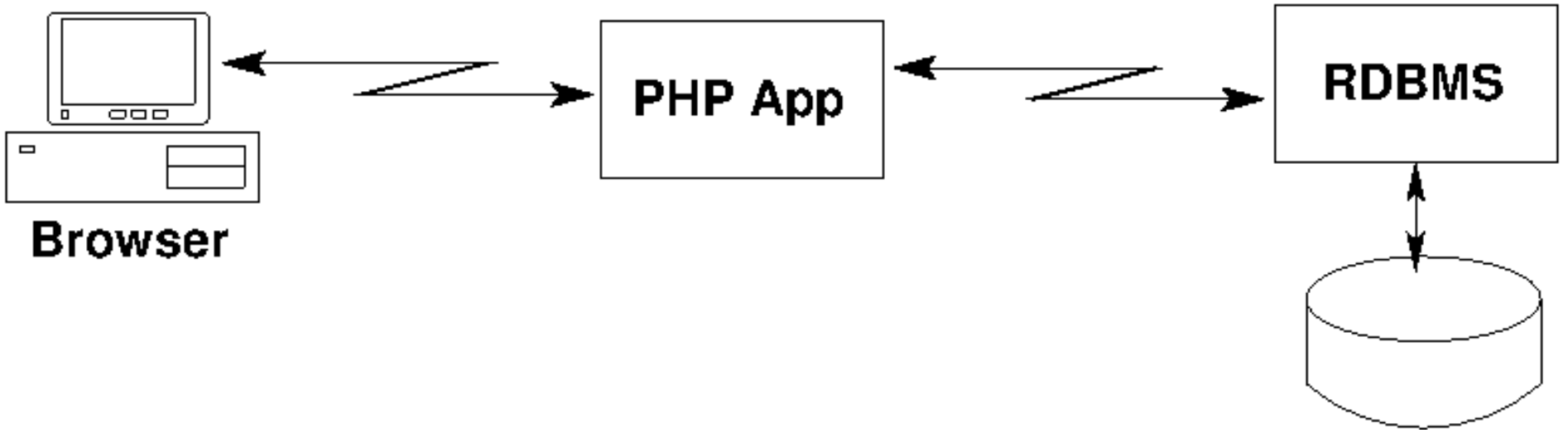
user interface, minimal plausibility checks

mixture of application logic, plausibility and integrity checks

used as pure data storage

How to do it better?

Typical PHP Application

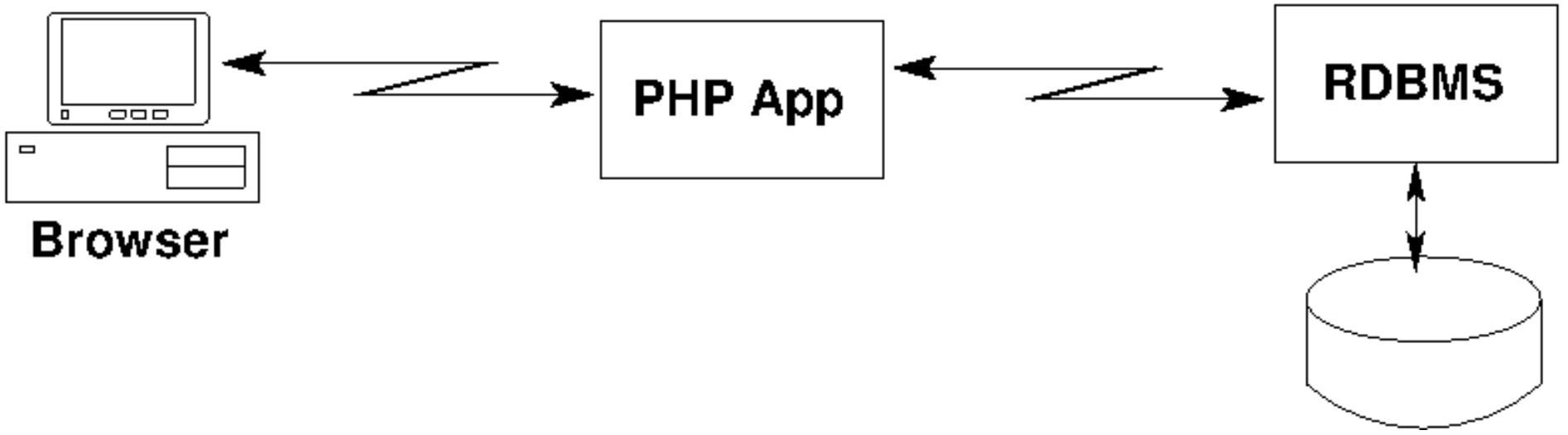


user interface, minimal plausibility checks

pure application logic, nothing else

data storage, integrity checks and some logic "behind the scenes"

Typical PHP Application



user interface, minimal plausibility checks

pure application logic, nothing else

data storage, integrity checks and some logic "behind the scenes"

Why is this better?

Integrity Checks => Database

Why to put **all** your integrity checks into the database:

- Declaring is easier than programming.
- Fast! And race conditions can be avoided.
- Consistent checks for all frontends
- Cannot be bypassed by interactive tools
- Smaller application programs
- Little checks in order to avoid network traffic remain allowed.

Why Do They Do It?

Why do people use SQL databases as pure means of saving data instead of making use of all the goodies in SQL?

- Many have “learnt” SQL by doing using some inferior system lacking many features.
- Some literature does not point out what is possible with SQL, because it concentrates on the application programming language.
- Some may use database frontends for generating the tables – and they don't offer more.

What does SQL offer regarding integrity checks?

- data types and domains
  RDBMS do not allow to save data not conforming to the data type – e. g. saving '2011-02-30' in a date field.
- check constraints
  Simple checks using only data from the current record.
- primary keys and unique constraints
  They make sure these columns or column groups are unique, e. g. unique employee id + day.
- foreign keys
  They make sure you don't insert data referencing something nonexistant, e. g. entering an order for a customer id which has already been deleted.

More helpful things

- Automatically numbering ids avoiding a “race condition” by using sequences/auto-increment.
- Giving users access only to views instead of real tables, thus limiting access.
- Doing complex integrity checks using triggers.
- Keeping a history of changes using triggers.
- Multi-user write access is flawless in most cases – no manual locks necessary!

SQL Data Types For Text

Most important data types include:

- `char(`*`max-length`*`)`
  use for character sequences (strings) of the same length, e. g. ISBN
- `varchar(`*`max-length`*`)`
  use for character sequences (strings) of variable length, e. g. names, descriptions
- `text`
  Attention: non-standard

SQL Data Types For Numbers

Most important data types include:

- `smallint, integer, bigint`
  2/4/8 bytes in size, positive/negative
- `numeric(`*`precision`*`,`*`scale`*`)`
  up to 1000 digits, precise, slow arithmetic
  indispensible for monetary values
- `real, double precision`
  real: 1E-37 to 1E+37, double: 1E-307 to 1E+308
  imprecise, fast arithmetic
  for measurements, maths, physics

SQL Data Types For Dates/Times

Most important data types include:

- `date`
  just a date, without time (not so in Oracle)
- `time`
  just a time, without date, microseconds
- `timestamp`
  date and time, with/without time zone, microseconds
- interval
  difference between timestamp values, duration

Other SQL Data Types

Other data types include:

- `boolean` just a bit
- `enumerated`
  user-defined using `create type`
- `geometric types`
  points, line segments, boxes, paths, polygons, circles
- network address types
  inet/cidr (IPv4 and IPv6), macaddr
- UUID, XML types, Arrays, Composite Types

Constraints

Allowing less than the data type would allow:

- `not null constraint` – disallowing null values
- `check constraint` – limit value, based only on the current record
- `create domain` – create your own data type including default value, `check` and `not null` constraints
- `foreign key constraint` – limit value based on something unique in another table

1,1 : 0,N Relationship

- An employee can have 0 or more (up to ∞) working hours records.
- A working hours record belongs to exactly one employee record.

emp_id	name	dateofbirth
1	Anna	1978-04-06
2	Fritz	1983-12-04
3	Berta	1980-11-11

emp_id	**day**	arrives	leaves	break
1	2011-08-16	08:00	17:30	0:30
1	2011-08-17	08:15	18:00	0:45
1	2011-08-18	07:55	16:00	0:45
2	2011-08-02	09:00	18:00	1:00

Remember

- Working hours referring to a nonexistant employee cannot be entered by mistake.
- Employees with working hours records cannot be deleted by mistake.
- If an employee id is changed (primary keys actually should never change), good RDBMS can change all referring records.
- This is called `ON UPDATE CASCADE`
- All this happens without any programming, fully automatically!

But what if ...

... things get more complicated?

You will have to write triggers. There you have to program, not just declare, right. But still:

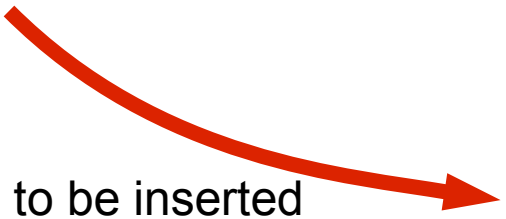
> Trigger programming has to be done **once** – not over and over again for every frontend.

Let's try a really complicated case:
A new working hours record may not be added if the previous one has not been completed by filling in the `leaves` field.

Entering a New Record

emp_id	name	dateofbirth
1	Anna	1978-04-06
2	Fritz	1983-12-04
3	Berta	1980-11-11

5	1	2011-08-19	08:15	0:00



new record to be inserted
OK or not?

should be rejected,
because this value is still NULL

emp_id	day	arrives	leaves	break
1	2011-08-16	08:00	17:30	0:30
1	2011-08-17	08:15	18:00	0:45
1	2011-08-18	07:55		0:45
2	2011-08-02	09:00	18:00	1:00

Entering a New Record

emp_id	name	dateofbirth
1	Anna	1978-04-06
2	Fritz	1983-12-04
3	Berta	1980-11-11

How can this rule be enforced without programming in the frontend?

should be rejected,
because this value is still NULL

5	1	2011-08-19	08:15	0:00

new record to be inserted
OK or not?

emp_id	day	arrives	leaves	break
1	2011-08-16	08:00	17:30	0:30
1	2011-08-17	08:15	18:00	0:45
1	2011-08-18	07:55		0:45
2	2011-08-02	09:00	18:00	1:00

Checking this Rule (1)

First a function has to be created:

```
create function check_left_yesterday() returns trigger as '
  declare lastleft time;
  begin
    select leaves into lastleft from working_hours
    where employee_id=new.employee_id
      and day = (select max(day) from working_hours
                 where employee_id=new.employee_id);
    if found and lastleft is null then
      raise exception ''% into table "%": field "leaves" of
            last entry is still NULL.'', TG_OP, TG_RELNAME;
    end if;
    return new;
  end;
' language 'plpgsql';
```

Checking this Rule (2)

Then a trigger can call this function:

```
create trigger check_left_yesterday
before insert on working_hours
for each row execute procedure check_left_yesterday();
```

Let's try it now:

```
insert into working_hours (employee_id, day, arrives)
                  values (1,'2011-08-19', '07:45');
ERROR:  INSERT into table "working_hours": field "leaves" of
last entry is still NULL.

update working_hours set leaves = '16:00'
  where employee_id=1 and day='2011-08-18';
UPDATE 1

insert into working_hours (employee_id, day, arrives)
                  values (1,'2011-08-19', '07:45');
INSERT 0 1
```

Is Everything Safe Now?

- Does this trigger make sure that there never is a missing `leaves` field?
- Nope! Only entering a new record is rejected if the last record lacks the `leaves` value.
- Still old `leaves` values can be set to `NULL` by updating existing records.
- Of course this can be avoided by creating a trigger for `UPDATE`.

Using Views

- Not everybody should see everything.
- Create views showing only some data and grant SELECT for certain users only for these views.
- Let's try to create a view showing only the added-up working hours for people.

View: accumulated working hours

Showing the accumulated working hours per employee:

```
create view sum_workinghours as
  select employee_id, name,
         sum(leaves-arrives-break) as hours
  from employee natural join working_hours
  group by employee_id, name;
```

Let's try it out:

```
select * from sum_workinghours;
 employee_id | name  |  hours
-------------+-------+----------
           2 | Fritz | 08:00:00
           1 | Anna  | 28:20:00
(2 rows)
```

View: accumulated working hours

Showing the accumulated working hours per employee:

```
create view sum_workinghours as
  select employee_id, name,
         sum(leaves-arrives-break) as hours
  from employee natural join working_hours
  group by employee_id, name;
```

sum up the hours minus the breaks

using both tables in a natural join

and group the results per employee

Let's try it out:

```
select * from sum_workinghours;
 employee_id | name  |  hours
-------------+-------+----------
           2 | Fritz | 08:00:00
           1 | Anna  | 28:20:00
(2 rows)
```

What's The Point?

What have we seen (maybe learnt) so far?

- You can keep everything regarding your data and their integrity in the RDBMS.
- Application programming can be freed from doing integrity checks leading to less cluttered code.
- Checks are more reliable and faster this way.
- Especially race conditions can be avoided.

How Do I Migrate To PostgreSQL? (1)

- Of course it's best to start a new project right away with a powerful RDBMS.
- There's a precondition for migrating existing applications to PostgreSQL: You have to be able to dump your database into SQL close to the standard.
- Otherwise, some search/replace or hand coding is necessary. Here more disadvantages of your previous database system may show.
- If so, this is not the fault of PostgreSQL.

How Do I Migrate To PostgreSQL? (2)

When you have dumped your old database into some SQL file, add schema management:

- `create schema humanresources`
  creates a schema (kind of directory) for tables, sequences, views and so on.
- `set search_path to humanresources`
  puts your subsequently created objects into the new schema.
- `drop schema humanresources`
  deletes the schema and all objects contained – in case you want to start over.

What Not To Do ...

Don't use several databases:

- You cannot use objects from several databases in one statement.
- You cannot enforce foreign key constraints spanning databases.

Don't use prefixes in table names:

- They are just a lame replacement for standard schemas

How Do I Migrate To PostgreSQL? (3)

Don't forget to remove all non-standard things from your file, like for instance:

- table types
- backticks `` `xx` `` and square brackets `[xx]`
- funny column types (like `datetime` or `varchar2`) and names (`user` is a reserved word)

Now let's import the definition of schemas, tables, sequences into PostgreSQL:

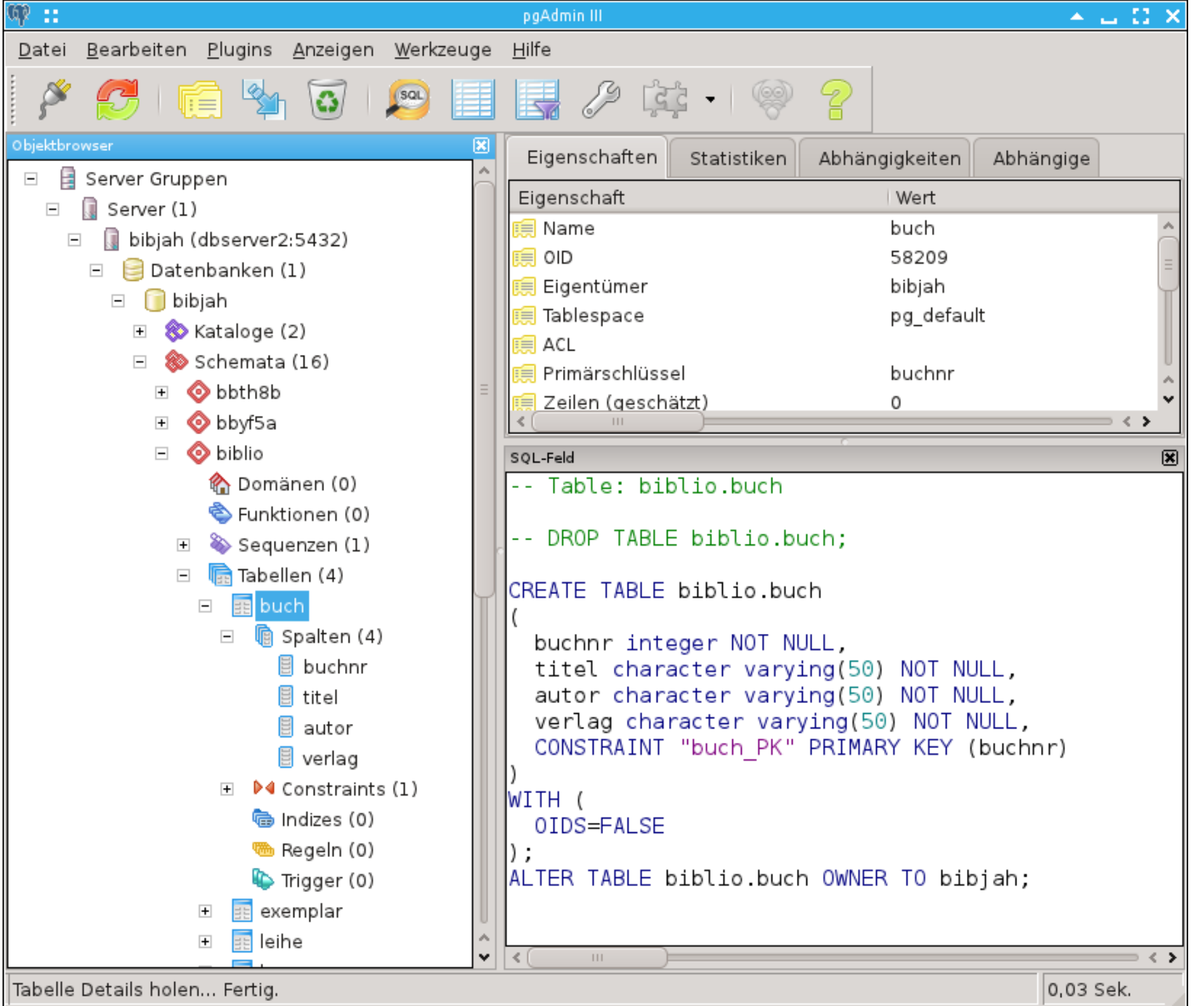
```
$ psql < db_definition.sql
```

Is PostgreSQL difficult?

- For everybody who as learnt SQL based on the ANSI standard, it's easy and straightforward.
- It does not mimic the quirks and deviations of other popular database systems and may seem awkward for those familiar with those.
- There are not many books. For a reason: The original documentation is already excellent!
- `psql` on the command line is great.
- `pgAdmin3` as a graphical tool is great as well.

Availablility of PostgreSQL

- It's included in the repositories of most Linux distributions and BSD.
- Binary packages are available for FreeBSD, several Linux distributions, Mac OS X, Solaris and Windows (32 and 64 bit)
- Live CDs and Software appliances:
  - PostgreSQL Live CD (Fedora and CendOS based)
  - TurnKey PostgresQL (Ubuntu based)
  - LAPP appliance and Amazon EC2 image


pgAdmin III
Datei Bearbeiten Plugins Anzeigen Werkzeuge Hilfe
Objektbrowser
Server Gruppen
Server (1)
bibjah (dbserver2:5432)
Datenbanken (1)
bibjah
Kataloge (2)
Schemata (16)
bbth8b
bbyf5a
biblio
Domänen (0)
Funktionen (0)
Sequenzen (1)
Tabellen (4)
buch
Spalten (4)
buchnr
titel
autor
verlag
Constraints (1)
Indizes (0)
Regeln (0)
Trigger (0)
exemplar
leihe
Eigenschaften
Statistiken
Abhängigkeiten
Abhängige
Eigenschaft
Wert
Name
buch
OID
58209
Eigentümer
bibjah
Tablespace
pg_default
ACL
Primärschlüssel
buchnr
Zeilen (geschätzt)
0
SQL-Feld
-- Table: biblio.buch
-- DROP TABLE biblio.buch;
CREATE TABLE biblio.buch
(
buchnr integer NOT NULL,
titel character varying(50) NOT NULL,
autor character varying(50) NOT NULL,
verlag character varying(50) NOT NULL,
CONSTRAINT "buch_PK" PRIMARY KEY (buchnr)
)
WITH (
OIDS=FALSE
);
ALTER TABLE biblio.buch OWNER TO bibjah;
Tabelle Details holen... Fertig.
0,03 Sek.