eZ systems

Welcome!

Share your information

eZ systems

Share your information

Contents

Design

- Table Design
- ER Diagrams
- Normalization
- Relations
- Intregity

Performance

- Indexes
- Denormalization
- Triggers

Tips & Tricks

- Nested Set
- Trees

eZ systems

Some Sample Data

- "Derick lives in Netherlands (NL) and works on Base 1.0, DatabaseSchema 1.0, File 1.0 and 1.1, Translation 1.0 and 1.1, UserInput 1.0 and 1.1"
- "Sergey lives in Ukraine (UA) and works on Base 1.1, Database 1.0 and 1.1, PersistentObject 1.1, SystemInformation 1.0"
- "Frederik lives in Norway (NO) and works on Database 1.0 and 1.1, Mail 1.0 and 1.1, PersistentObject 1.0"

eZ systems

Share your information

First Attempt

Components	Name	C. Code	Country
Base 1.0	Derick	NL	Netherlands
DatabaseSchema 1.0	Derick	NL	Netherlands
File 1.0, File 1.1	Derick	NL	Netherlands
Translation 1.0, Translation 1.1	Derick	NL	Netherlands
UserInput 1.0, UserInput 1.1	Derick	NL	Netherlands
Base 1.1	Sergey	UA	Ukraine
Database 1.0, Database 1.1	Sergey	UA	Ukraine
PersistentObject 1.1	Sergey	UA	Ukraine
SystemInformation 1.0	Sergey	UA	Ukraine
Database 1.0, Database 1.1	Frederik	NO	Norway
Mail 1.0, Mail 1.1	Frederik	NO	Norway
PersistentObject 1.0			

eZ systems

First Normal Form

All values in each column of a table are atomic. This means that there are no sets of values within a column.

Share your information

eZ systems

Second Attempt

Component	Name	C. Code	Country
Base 1.0	Derick	NL	Netherlands
DatabaseSchema 1.0	Derick	NL	Netherlands
File 1.0	Derick	NL	Netherlands
File 1.1	Derick	NL	Netherlands
Translation 1.0	Derick	NL	Netherlands
Translation 1.1	Derick	NL	Netherlands
UserInput 1.0	Derick	NL	Netherlands
UserInput 1.1	Derick	NL	Netherlands
Base 1.1	Sergey	UA	Ukraine
Database 1.0	Sergey	UA	Ukraine
...	...	...	...

- You can not store developers without component.
- Required to update multiple records when somebody moves.

Share your information

Second Normal Form

Any non-key columns must depend on the entire primary key. In the case of a composite primary key, this means that a non-key column cannot depend on only part of the composite key.

eZ systems

Primary Keys

2NF: "Any non-key columns must depend on the entire primary key..."

Component	Name	C. Code	Country
Base 1.0	Derick	NL	Netherlands
File 1.0	Derick	NL	Netherlands
File 1.1	Derick	NL	Netherlands

A primary key is a value that can be used to identify a unique row in a table.

Component	Name	C. Code	Country
Base 1.0	Derick	NL	Netherlands
File 1.0	Derick	NL	Netherlands
File 1.1	Derick	NL	Netherlands

eZ systems

Third Attempt

2NF: "Any non-key columns must depend on the entire primary key. In the case of a composite primary key, this means that a non-key column cannot depend on only part of the composite key."

Component	Name	C. Code	Country
Base 1.0	Derick	NL	Netherlands
DatabaseSchema 1.0	Derick	NL	Netherlands
File 1.0	Derick	NL	Netherlands
Base 1.1	Sergey	UA	Ukraine
Database 1.0	Sergey	UA	Ukraine
...	...	...	...

Name	C. Code	Country
Derick	NL	Netherlands
Frederik	NO	Norway
Sergey	UA	Ukraine

Share your information

eZ systems

Third Normal Form

All columns must depend directly on the primary key.

Share your information

eZ systems

Fourth Attempt

3NF: "All columns must depend directly on the primary key."

Name	C. Code	Country
Derick	NL	Netherlands
Frederik	NO	Norway
Raymond	NL	Netherlands
Sergey	UA	Ukraine

C. Code	Country
NL	Netherlands
NO	Norway
UA	Ukraine

Share your information

eZ systems

First Normal Form Revised

1NF: "All values in each column of a table are atomic. This means that there are no sets of values within a column."

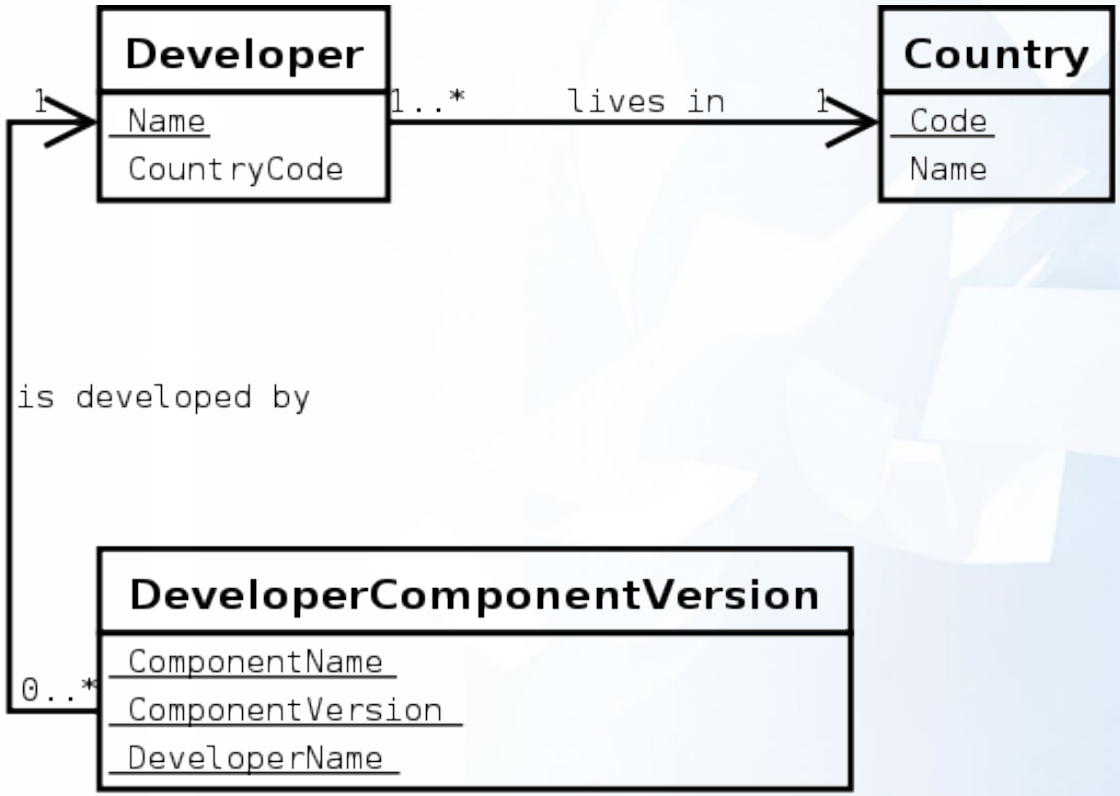
Component	Name
Base 1.0	Derick
DatabaseSchema 1.0	Derick
File 1.0	Derick
...	...

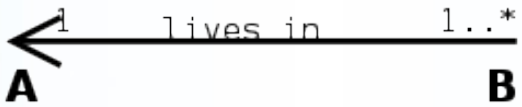
Component	Component Version	Name
Base	1.0	Derick
DatabaseSchema	1.0	Derick
File	1.0	Derick
...	...	...

Share your information

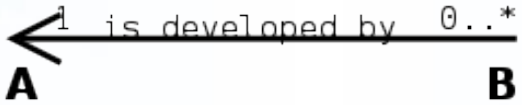
eZ systems

Entity Relationship Diagram





Entity A has one or more rows in entity B.
A Country can be associated with one or more Developers - this also means that we do not store Countries where we have no developer living.



Entity A has zero or more rows in entity B.
A Developer can be assigned to zero or more Components - this also means that we do store Developers which have nothing to do.

Share your information

eZ systems

Primary Key Mangling

Component	Component Version	Developer Name
Base	1.0	Derick
Base	1.1	Sergey

Component	Component Version	Release Date
Base	1.0	2006-01-07
Base	1.1	NULL

to:

Component Version ID	Developer Name
1	Derick
3	Sergey

ID	Component	Component Version	Release Date
1	Base	1.0	2006-01-07
3	Base	1.1	NULL

Share your information

eZ systems

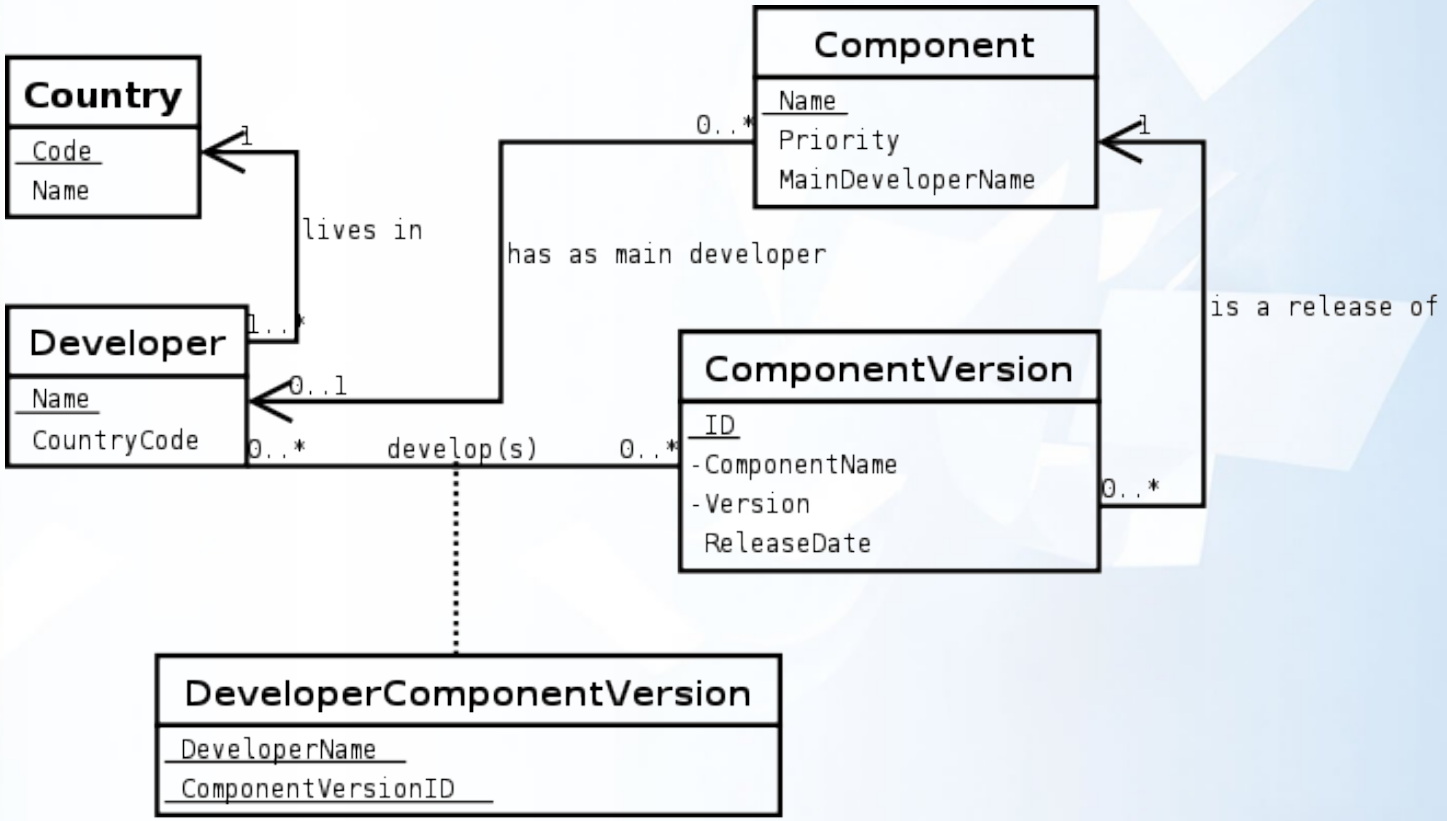
Primary Key Mangling

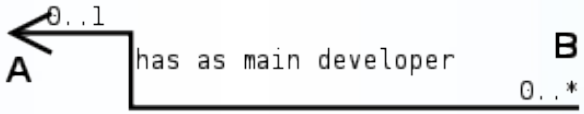
ID	Component	Component Version	Release Date
1	Base	1.0	2006-01-07
3	Base	1.1	NULL

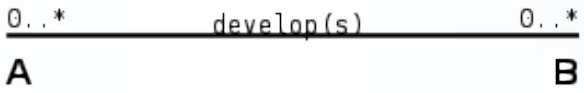
- Component, Component Version was the original primary key and still identifies each record uniquely.
- A (combination of) key(s) that identify a record uniquely is called a candidate key.
- One of the candidate keys can be defined as the primary key (ID in the example above).
- ID is an artificial key (surrogate key).
- A surrogate key is not always required, but can increase performance while joining or selecting.

eZ systems

Entity Relationship Diagram







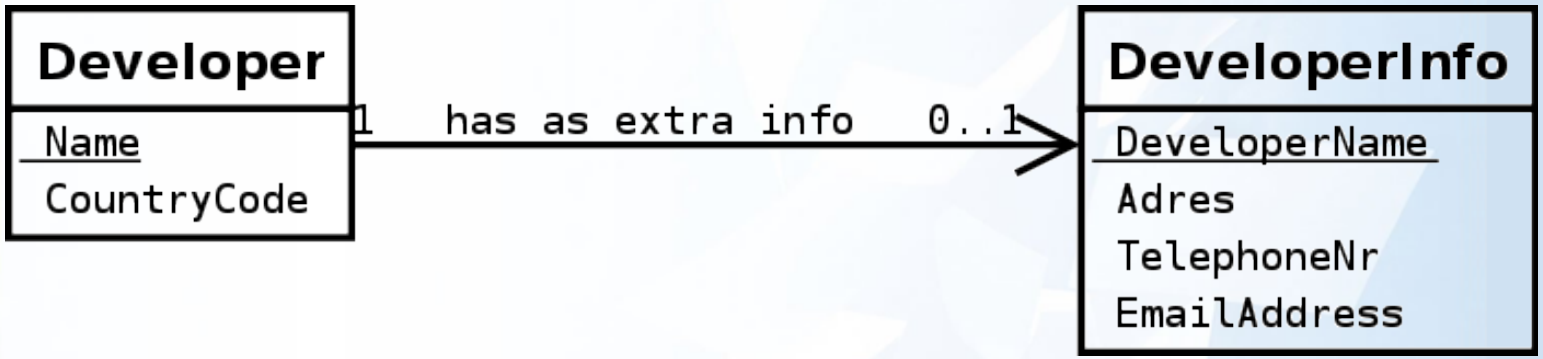
Entity A has zero or more rows in entity B.
A Developer can be the main developer of zero or more Components. A Component can optionally have a Developer as main developer.

Entity A has a many to many relation with entity B.
A ComponentVersion can be developed by multiple developers and one developer can work on multiple ComponentVersions. Can not be done in Relational Databases.

Share your information

Relations

Usually does not need a seperate table. But can be useful to save space:



eZ systems

Relations

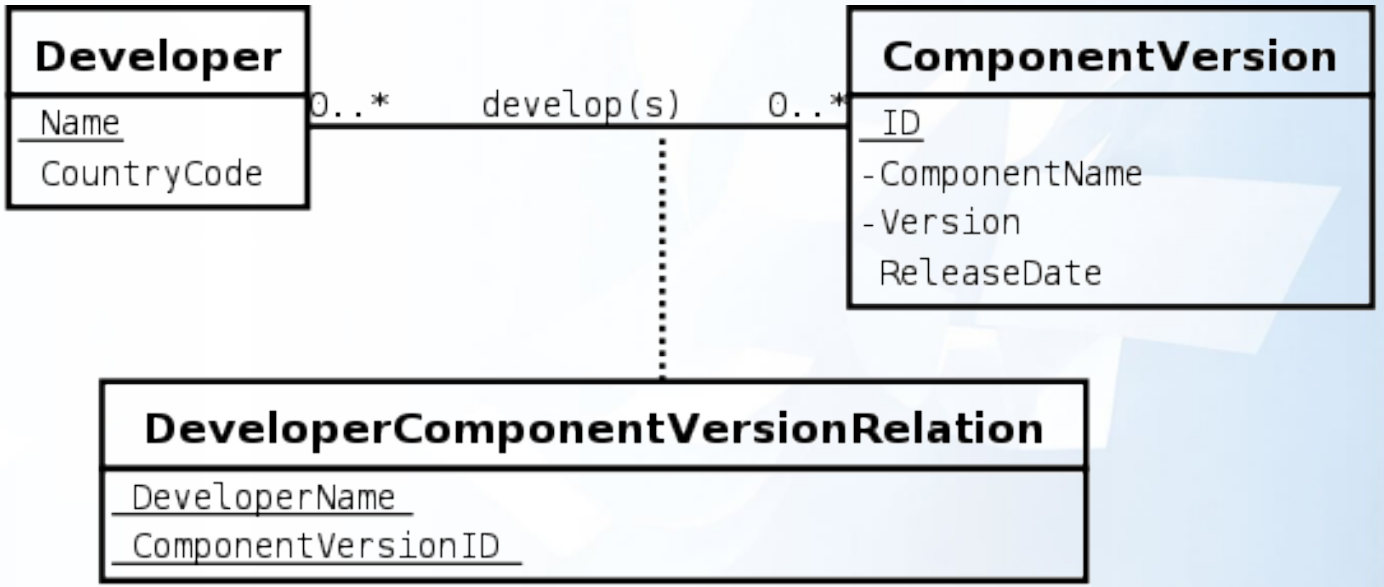
Most frequently used type of relation:

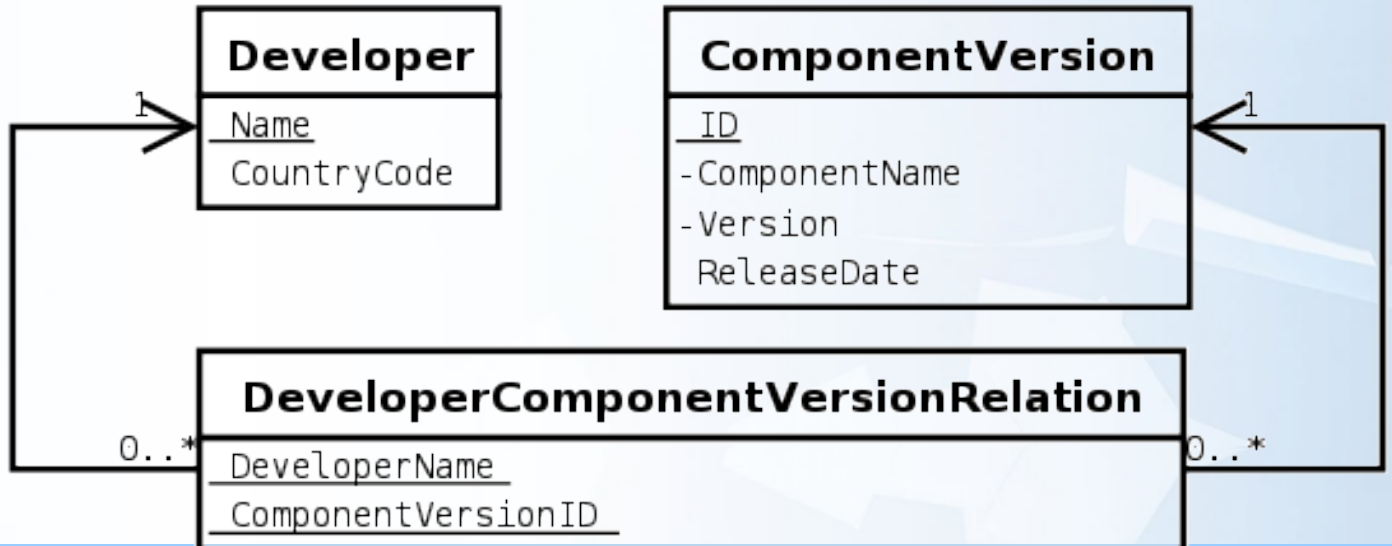
Share your information

eZ systems

Relations

You need a special table to record the relation itself:





Share your information

eZ systems

Naming Tables

Two major options:

- Singular names: Component, ComponentVersion, Developer, Country
- Plural names: Components, ComponentVersions, Developers, Countries

Naming "connection tables" for Many to Many relations:

- Include the names of both tables into the name (Developer + ComponentVersion -> DeveloperComponentVersion)
- Optionally add Relation to this name (Developer + ComponentVersion -> DeveloperComponentVersionRelation)

Naming Columns

Normal columns are straight forward.

For columns that refer to another table, it is useful to:

- Prefix them with the table name they refer to.
- Use the same name as the column name in the table it is referred to.

eZ systems

Share your information

Assignment #1

Design a database (as ERD) to store the following data:

- Derick has a return ticket on Monday April 24th, 2006 from TRF (Torp) to MCO (Orlando) and back on April 29th
- The flight trom Torp to Orlando consists of: TRF to AMS (Amsterdam), AMS to DTW (Detroit) and DTW to MCO, the way back goes MCO->MEM (Memphis)->AMS->TRF
- Flight KL 1212 leaves every day at 06:25 and goes from TRF to AMS
- Flight KL 1221 leaves on all days, except Saturday at 15:30 from AMS to TRF
- Flight KL 1212 and KL 1221 takes 1h40m
- Derick has a single ticket on Saturday May 13th from AMS to TRF.

Break

Column Types

A column can have a different datatype. Datatypes are often RDBMS dependent, but there are often equivalent types with a slightly different name.

The following types are often available:

- integer: to store integral numbers
- float: to store floating point values
- varchar: to store variable length strings
- char: to store fixed length strings
- date: to store dates
- timestamp: to store timestamps
- blob: to store (large) amounts binary data
- clob or text: to store (large) amounts of text

eZ systems

NULL

- "NULL" is a special value
- "NULL" is neither TRUE or FALSE
- "NULL = NULL" evaluates to FALSE
- "NULL IS NULL" evaluates to TRUE
- Columns can be defined as "NULL" (default) or "NOT NULL"

Share your information

eZ systems

Auto Increment

- Generates the 'next' number upon insert
- Often used for surrogate keys
- In some database not available directly, but only through a sequence
- Column must defined as "integer not null"
- They can not be cleaned up

Share your information

eZ systems

Share your information

Auto Increment

The following has been editted to fix spelling errors and to protect the guilty:

```
user: I'm having a problem with my MySQL database :)
user: I have a Table with Field1, which has auto_increment attribute
user: Table has three records, with Field1 numbered 1, 2 and 3
user: now I'm using delete from Table where Field1 = 2 limit 1;
user: and next id is 4..
user: but I want it to use 2
user: I read something about resetting auto_increment.. is that
    the way to fix it then?
user: I'll run out of id's otherwise
user: but the thing is, this company wants to add a lot of
    pictures to their database
user: because selling pictures is what they do :)
user: now I made Field1 unsigned int, but what if it turns out to
    be not enough?
user: but, just out of curiosity.. is there a way to fill up gaps
    with auto_increment? :)
```

- unsigned int is enough!
- for unsigned int the company has to hire 20 photographers which have to shoot 1000 pictures per day. Then they have to work 588 years until you run out

eZ systems

CREATE TABLE

```
+------+-------------+
| Code | Name        |
+------+-------------+
| NL   | Netherlands |
+------+-------------+
| NO   | Norway      |
+------+-------------+
```

```
CREATE TABLE country (
    code CHAR(2) NOT NULL,
    name VARCHAR(64) NOT NULL,
    PRIMARY KEY (code)
) ENGINE=InnoDB;
```

```
+--------+--------------+
| Name   | Country Code |
+--------+--------------+
| Derick | NL           |
+--------+--------------+
| Sergey | NO           |
+--------+--------------+
```

```
CREATE TABLE developer (
    name VARCHAR(64) NOT NULL,
    country_code CHAR(2) NOT NULL,
    PRIMARY KEY (name)
) ENGINE=InnoDB;
```

Share your information

eZ systems

CREATE TABLE

```
+----------------+----------+----------------------+
| Name           | Priority | Main Developer Name |
+----------------+----------+----------------------+
| DatabaseSchema | 2        | Derick               |
+----------------+----------+----------------------+
| Database       | 3        | NULL                 |
+----------------+----------+----------------------+
```

```
CREATE TABLE component (
    name VARCHAR(64) NOT NULL,
    priority INT NOT NULL,
    main_developer_name VARCHAR(64),
    PRIMARY KEY (name)
) ENGINE=InnoDB;
```

Share your information

eZ systems

CREATE TABLE

```
+----+-----------+-------------------+--------------+
| ID | Component | Component Version | Release Date |
+----+-----------+-------------------+--------------+
| 1  | Base      | 1.0               | 2006-01-07   |
+----+-----------+-------------------+--------------+
| 3  | Base      | 1.1               | NULL         |
+----+-----------+-------------------+--------------+
```

```
CREATE TABLE component_version (
    id INT NOT NULL AUTO_INCREMENT,
    component_name VARCHAR(64) NOT NULL,
    component_version VARCHAR(16) NOT NULL,
    release_date date,
    PRIMARY KEY (id),
    UNIQUE KEY name_version (component_name, component_version)
) ENGINE=InnoDB;
```

```
+----------------------+----------------+
| Component Version ID | Developer Name |
+----------------------+----------------+
| 1                    | Derick         |
+----------------------+----------------+
| 3                    | Sergey         |
+----------------------+----------------+
```

```
CREATE TABLE developer_component_version (
    component_version_id INTEGER NOT NULL,
    developer_name VARCHAR(64) NOT NULL,
    PRIMARY KEY (component_version_id, developer_name)
) ENGINE=InnoDB;
```

Share your information

eZ systems

Intregity

- In our current database there is no intregity checking
- Data in a table's record might refer to another table without that data being present: inconsistent data
- Referential Intregity

Share your information

eZ systems

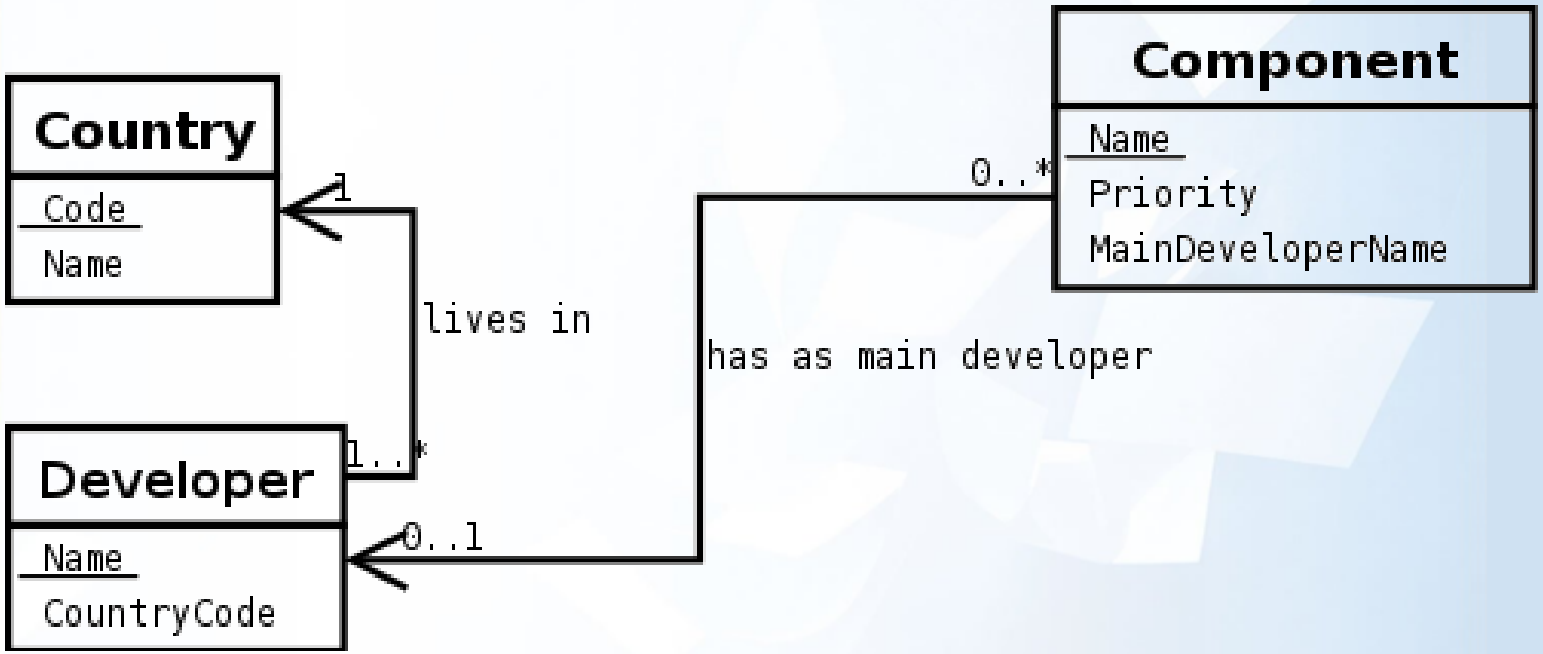
Foreign Keys

- Foreign keys act as constraints to guard against data inconsistency
- They make sure that data which is refered to is actually there
- Can be set-up in such a way so that refered data is automatically removed

Share your information

eZ systems

Adding the Foreign Keys



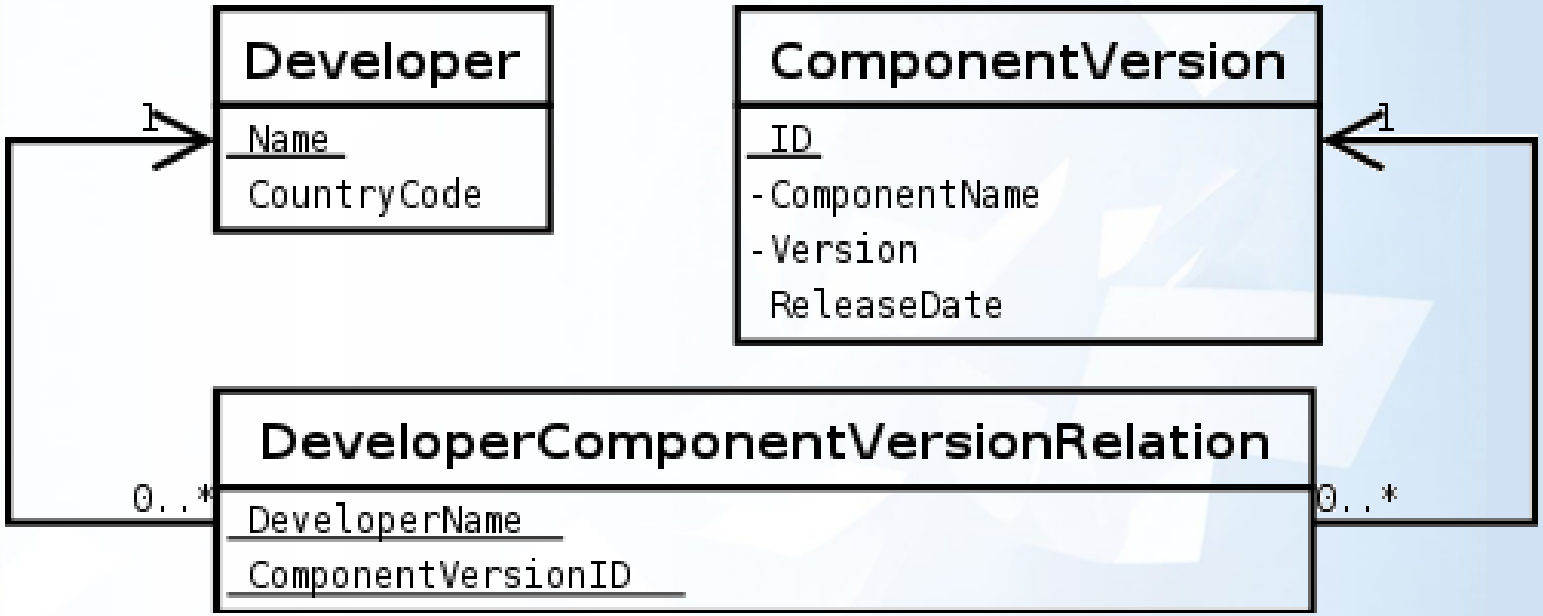

```
ALTER TABLE developer
    ADD CONSTRAINT developer_country_code
    FOREIGN KEY (country_code) REFERENCES country(code);

ALTER TABLE component
    ADD CONSTRAINT component_developer_name
    FOREIGN KEY (main_developer_name) REFERENCES developer(name);
```

Share your information

eZ systems

Adding the Foreign Keys



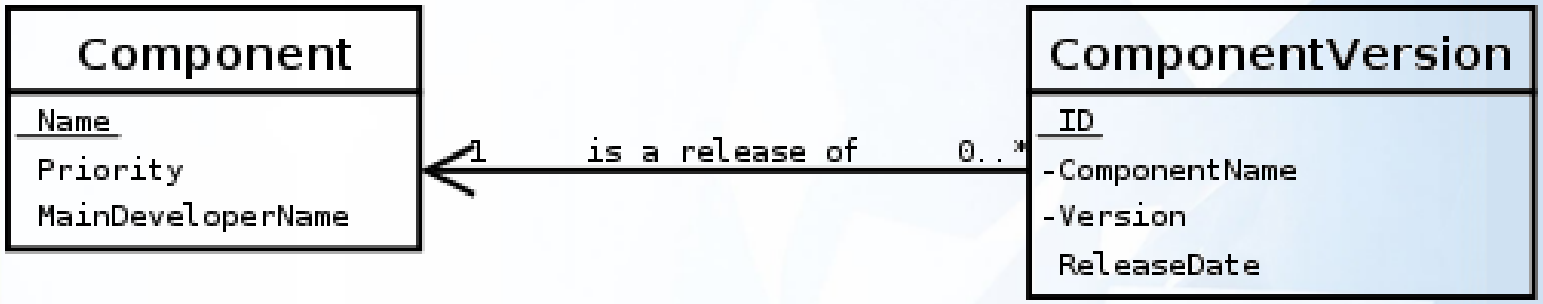

```
ALTER TABLE developer_component_version
    ADD CONSTRAINT developer_component_version_developer
    FOREIGN KEY (developer_name)
        REFERENCES developer(name),

    ADD CONSTRAINT developer_component_version_component_version
    FOREIGN KEY (component_version_id)
        REFERENCES component_version(id);
```

Share your information

eZ systems

Adding the Foreign Keys




```
ALTER TABLE component_version
    ADD CONSTRAINT component_version_component
    FOREIGN KEY (component_name)
        REFERENCES component(name)
        ON DELETE CASCADE
        ON UPDATE CASCADE;
```

- ON DELETE CASCADE: If a by component_version referenced value is delete from component, also delete all the corresponding records in this table
- ON UPDATE CASCADE: If a by component_version referenced value in component changes due to an UPDATE query, update the values in component_version too

Share your information

eZ systems

Indexes

- An index orders data for fast retrieval
- Indexes can be used for querying or sorting
- Indexes also use memory

```
+----+-----------+-------------------+--------------+
| ID | Component | Component Version | Release Date |
+----+-----------+-------------------+--------------+
| 1  | Base      | 1.0               | 2006-01-07   |
+----+-----------+-------------------+--------------+
| 3  | Base      | 1.1               | NULL         |
+----+-----------+-------------------+--------------+
```

Searching for all releases in Q1 does not use an index:

```
EXPLAIN SELECT * FROM component_version
    WHERE release_date BETWEEN '2006-01-01' AND '2006-03-31';
```

Create an index with:

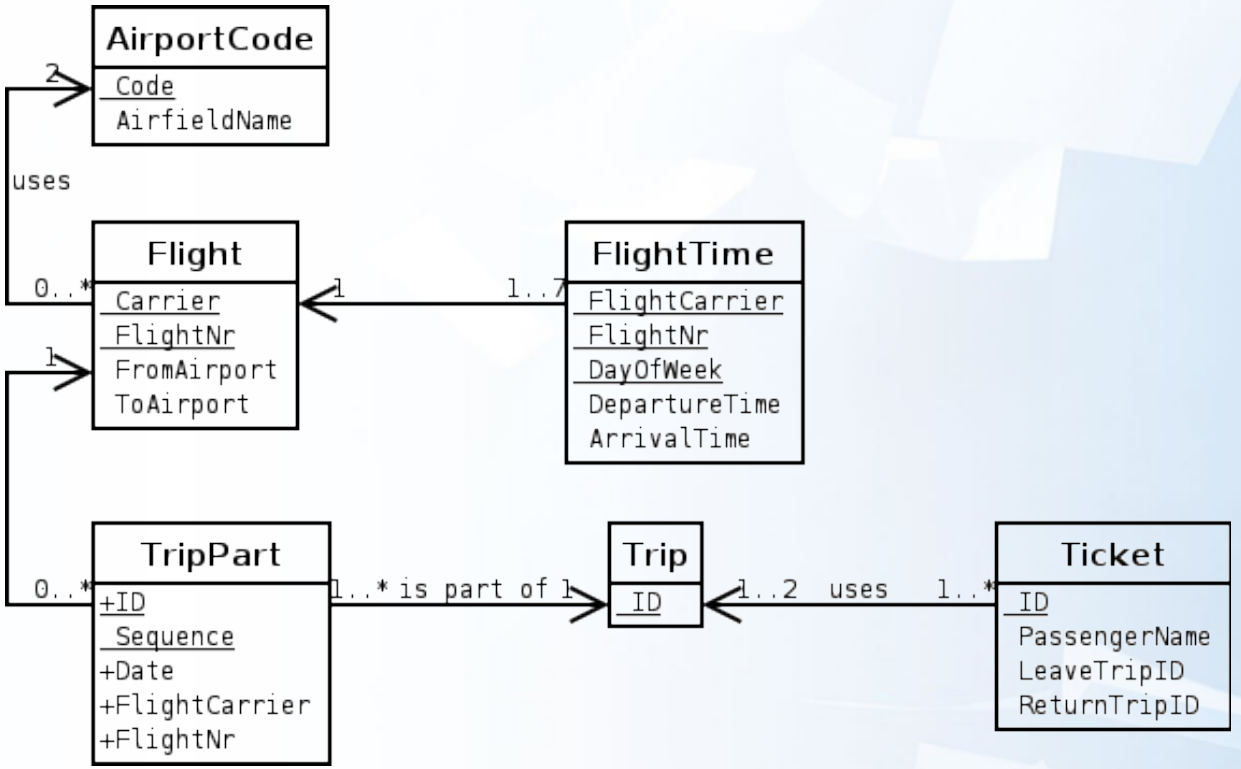
```
ALTER TABLE component_version
    ADD INDEX component_version_release_date (release_date);
```

Share your information

eZ systems

Assignment #2

Write down (or try with MySQL) the SQL (including possible useful indexes and constraints) of the flight and airport_code tables from the following diagram:



Share your information

Break

eZ systems

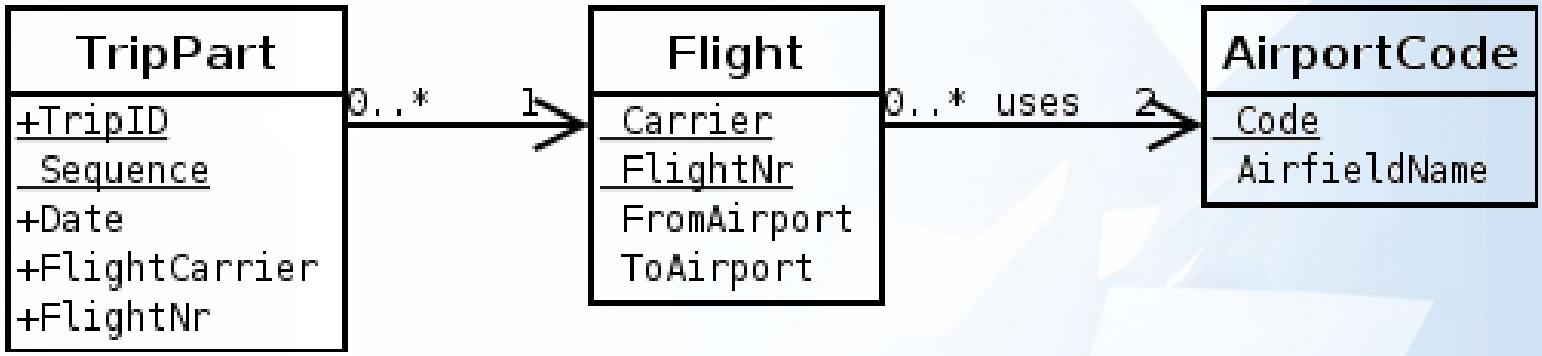
Denormalization

- Denormalization is the process of reverting a database schema to a lower NF
- Denormalization is often done to enhance performance

Share your information

eZ systems

Denormalization



Usually you always want to display the name of the airfield instead of the airport code. But in the current diagram you always have to use a join for this:

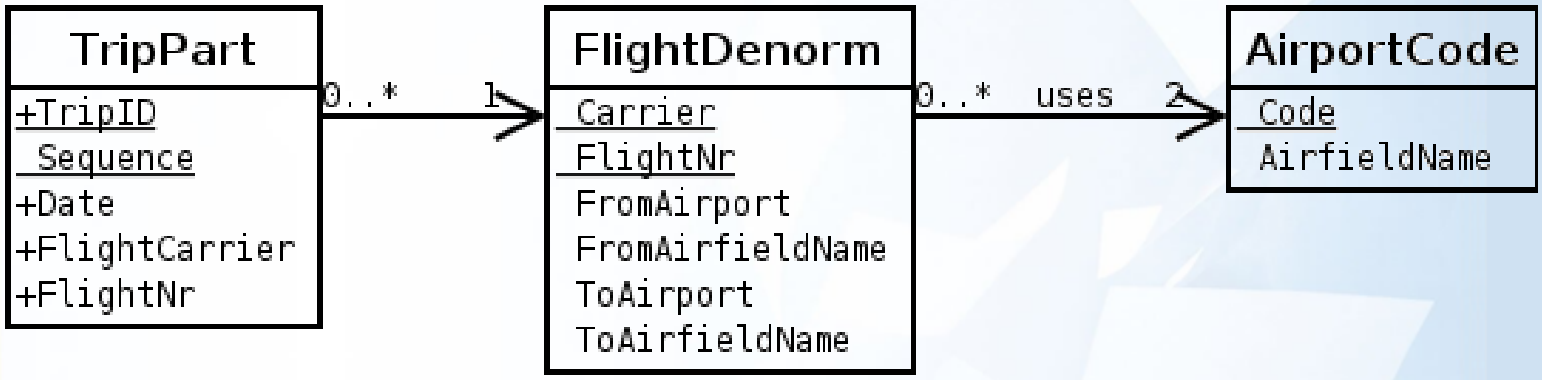
```
SELECT tp.date, tp.sequence, fa.airfield_name, ta.airfield_name
FROM trip_part tp
    JOIN flight f ON
    tp.flight_carrier = f.carrier AND tp.flight_nr = f.flight_nr
    JOIN airport_code fa ON f.from_airport = fa.code
    JOIN airport_code ta ON f.to_airport = ta.code
WHERE tp.trip_id = 1
ORDER BY tp.sequence;
```

```
+----------+--------+--------------+--------------+
```

Share your information

eZ systems

Denormalization



Usually you always want to display the name of the airfield instead of the airport code. But in the current diagram you always have to use a join for this:

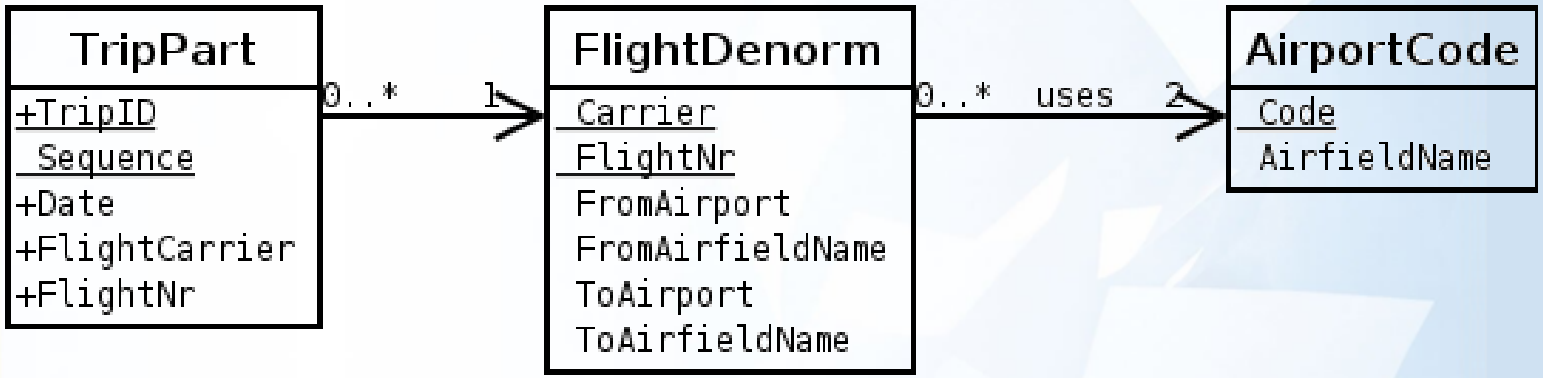
```
SELECT tp.date, tp.sequence,
    fd.from_airfield_name, fd.to_airfield_name
FROM trip_part tp
    JOIN flight_denorm fd ON
    tp.flight_carrier = fd.carrier AND tp.flight_nr = fd.flight_nr
WHERE tp.trip_id = 1
ORDER BY tp.sequence;


+------------+----------+--------------------+------------------+
| date       | sequence | from_airfield_name | to_airfield_name |
```

Share your information

eZ systems

Denormalization



If some airport changes name, you need to run multiple updates:

```
UPDATE airport_code
    SET airfield_name = 'Montreal Pierre Trudeau'
    WHERE code = 'YUL';

UPDATE flight_denorm
    SET from_airfield_name = 'Montreal Pierre Trudeau'
    WHERE from_airport = 'YUL';

UPDATE flight_denorm
    SET to_airfield_name = 'Montreal Pierre Trudeau'
    WHERE to_airport = 'YUL';
```

Share your information

eZ systems

Trigger

A better solution is a trigger:

Share your information

Tips & Tricks

Share your information

eZ systems

MySQL Table Types

MyISAM

- Efficient For either High Volume writes or reads
- Table level locking
- No Transaction support

InnoDB

- Efficient locking (Row-level update, non-locking read)
- Foreign Keys, Transactions
- High Concurrency

eZ systems

Trees

Trees do not really map well to relational databases. The most obvious way of storing them is like:

ID	Parent ID	Name
3	1	Countries
4	3	Belgium
5	3	Netherlands
6	3	Germany
7	5	Business
8	5	Economy

eZ systems

Trees

ID	Parent ID	Name
3	1	Countries
4	3	Belgium
5	3	Netherlands
6	3	Germany
7	5	Business
8	5	Economy

Retrieving the full path to the root node goes with something like:

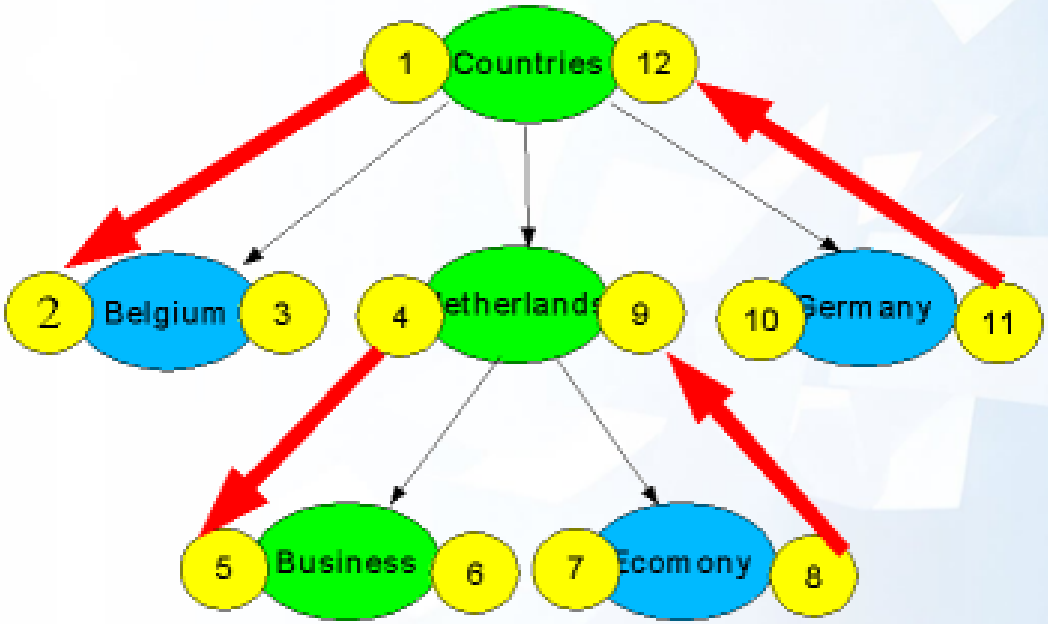
```
<?php
function collectPath( $id ) {
    global $dir;

    $res = mysql_query( "SELECT parent, name FROM dir WHERE id=$id" );
    if ( mysql_num_rows( $res ) > 0 ) {
        $dir[] = ( $row = mysql_fetch_row( $res ) );
        collectPath( $row['parent'] );
    }
}
$dir = array();
collectPath(7);
```

Share your information

eZ systems

Nested Set



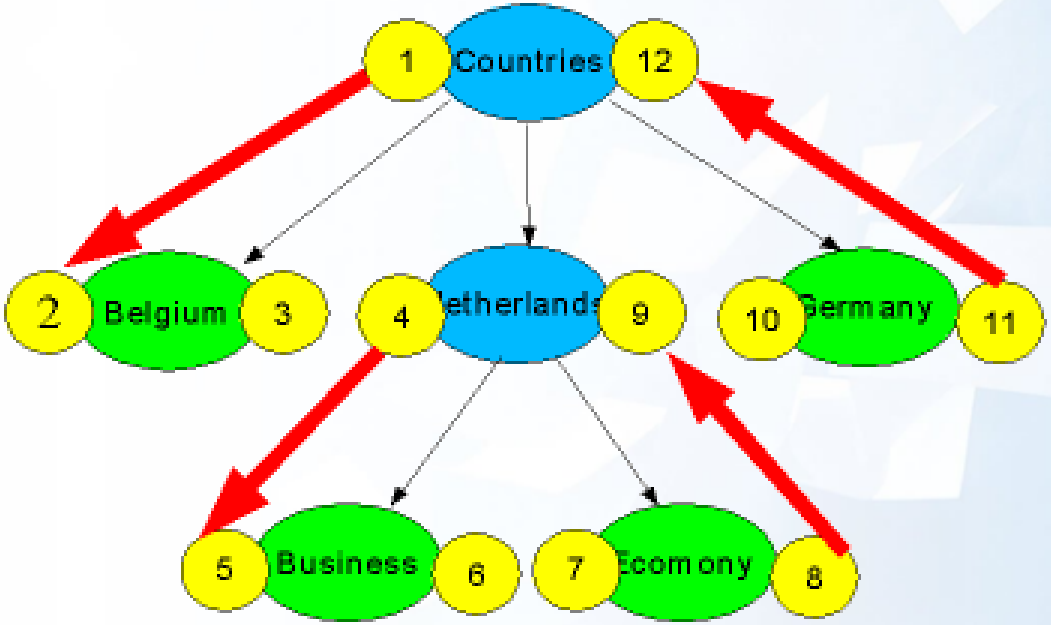
Query:

```
SELECT * FROM directory
WHERE
    left <= 5 AND right >= 6
```

Share your information

eZ systems

Nested Set



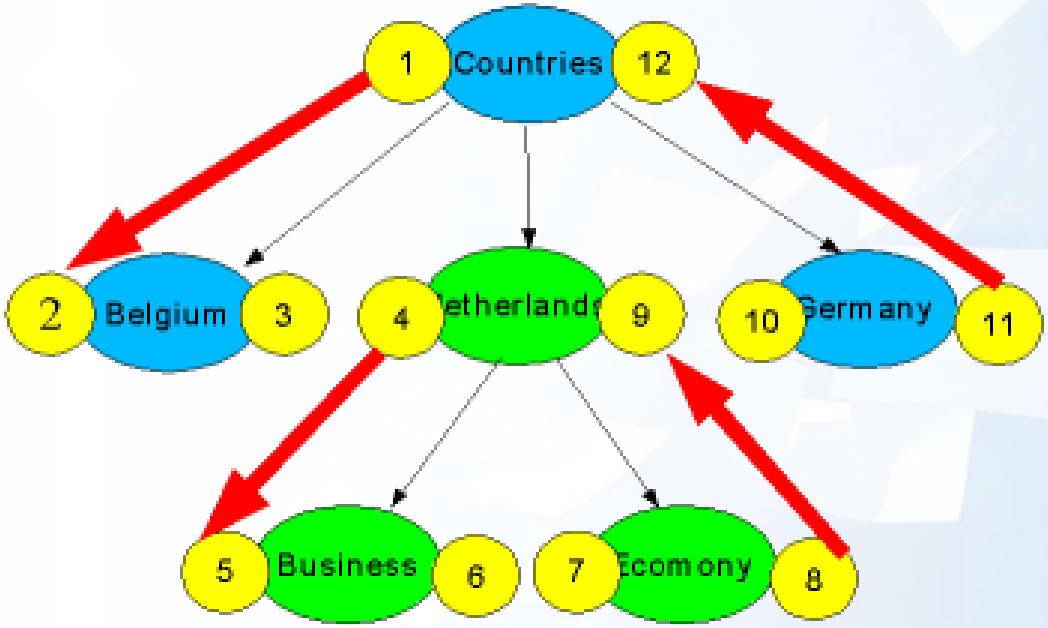
Query:

```
SELECT * FROM directory
WHERE
    right - left = 1
```

Share your information

eZ systems

Nested Set



Query:

```
SELECT * FROM directory
WHERE
    left >= 4 AND right <= 9
```

Share your information

eZ systems

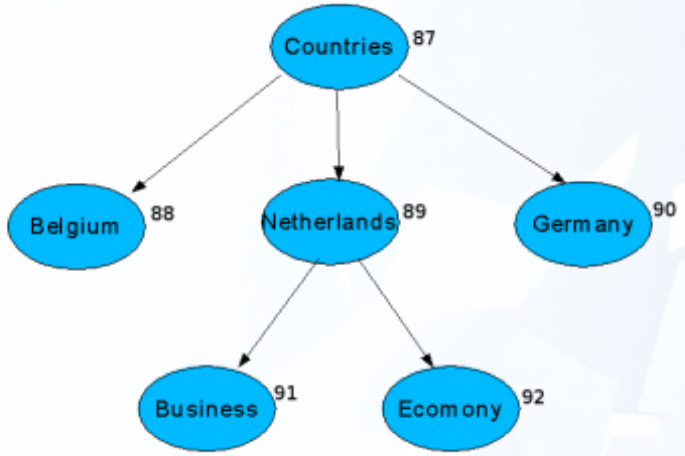
Nested Set

- Fast for reads and selection
- Doesn't scale well with adding nodes
- Doesn't scale well with moving nodes/subtrees

Share your information

eZ systems

Different Approach



Path strings:

```
Id | ParentId | PathString | Name         | Depth
---+----------+------------+--------------+------
87 |        2 | /87/       | Countries    | 2
88 |       87 | /87/88/    | Belgium      | 3
89 |       87 | /87/89/    | Netherlands  | 3
91 |       89 | /87/89/91/ | Business     | 4
92 |       89 | /87/89/92/ | Economy      | 4
90 |       87 | /87/90/    | Germany      | 3
```

Share your information

eZ systems

Share your information

Path Strings

```
Id | ParentId | PathString | Name          | Depth
---+----------+------------+---------------+------
87 |        2 | /87/       | Countries     | 2
88 |       87 | /87/88/    | Belgium       | 3
89 |       87 | /87/89/    | Netherlands   | 3
91 |       89 | /87/89/91/ | Business      | 4
92 |       89 | /87/89/92/ | Economy       | 4
90 |       87 | /87/90/    | Germany       | 3
```

Adding a node:

```
function add_node( parent, name )
{
    $parent_info =
        SELECT *
        FROM tree
        WHERE Id = $parent;

    INSERT INTO tree(Id, Name) VALUES( NULL, $name );

    $last_id = LAST INSERT ID;
    UPDATE tree
        SET
            ParentId = $parent,
            PathString = $parent_info['PathString'] . $last_id . "/",
            Depth = $parent['Depth'] + 1
        WHERE Id = $last_id
```

eZ systems

Path Strings

```
Id | ParentId | PathString | Name          | Depth
---+----------+------------+---------------+------
87 |        2 | /87/       | Countries     | 2
88 |       87 | /87/88/    | Belgium       | 3
89 |       87 | /87/89/    | Netherlands   | 3
91 |       89 | /87/89/91/ | Business      | 4
92 |       89 | /87/89/92/ | Economy       | 4
90 |       87 | /87/90/    | Germany       | 3
```

Selecting the path:

```
function select_path( id )
{
    $info =
        SELECT *
        FROM tree
        WHERE Id = $id;

    $parts =
        array_slice( split( '/', $info['PathString'] ), 1, -1 );

    $path =
        SELECT Id, Name
        FROM tree
        WHERE Id in $parts;

    return $path;
```

Share your information

eZ systems

Path Strings

```
Id | ParentId | PathString | Name          | Depth
---+----------+------------+---------------+------
87 |        2 | /87/       | Countries     | 2
88 |       87 | /87/88/    | Belgium       | 3
89 |       87 | /87/89/    | Netherlands   | 3
91 |       89 | /87/89/91/ | Business      | 4
92 |       89 | /87/89/92/ | Economy       | 4
90 |       87 | /87/90/    | Germany       | 3
```

Selecting a subtree:

```
function select_subtree( id )
{
    $info =
        SELECT PathString
        FROM tree
        WHERE Id = $id;

    $tree_elements =
        SELECT *
        FROM tree
        WHERE PathString LIKE "$info%";

    return $tree_elements;
}
```

Share your information

eZ systems

Questions and Resources

Questions anybody?

Resources:

- These Slides: http://files.derickrethans.nl/db-phptek6.pdf

Share your information