Normalization

DATABASE SYSTEM
SESSION 17

Normalization

- The biggest problem needed to be solved in database is data redundancy.
- Why data **redundancy** is the problem? Because it causes:
  - Insert Anomaly
  - Update Anomaly
  - Delete Anomaly

Teacher	Subject	Teacher Degree	Tel
Sok San	Database	Master's	012666777
Van Sokhen	Database	Bachelor's	017678678
Sok San	E-Commerce	Master's	012666777

Normalization (Cont.)

- Normalization is the process of removing redundant data from your tables to improve storage efficiency, data integrity, and scalability.
- Normalization generally involves splitting existing tables into multiple ones, which must be re-joined or linked each time a query is issued.
- Why normalization?
  - The relation derived from the user view or data store will most likely be unnormalized.
  - The problem usually happens when an existing system uses unstructured file, e.g. in MS Excel.

Steps of Normalization

- First Normal Form (1NF)
- Second Normal Form (2NF)
- Third Normal Form (3NF)
- Boyce-Codd Normal Form (BCNF)

First Normal Form (1NF)

The official qualifications for 1NF are:

1. Each **attribute name** must be unique.
2. Each **attribute value** must be single.
3. Each **row** must be unique.
4. There is **no repeating groups**.

- Additional:
  - Choose a primary key.

- Reminder:

A primary key is ***unique***, ***not null***, ***unchanged***. A primary key can be either an attribute or combined attributes.

First Normal Form (1NF) (Cont.)

- Example of a table not in 1NF :

Group	Topic	Student	Score
Group A	Intro MongoDB	Sok San	18 marks
		Sao Ry	17 marks
Group B	Intro MySQL	Chan Tina	19 marks
		Tith Sophea	16 marks

It violates the 1NF because:

- Attribute values are not single.
- Repeating groups exists.

First Normal Form (1NF) (Cont.)

- After eliminating:

Group	Topic	Family Name	Given Name	Score
A	Intro MongoDB	Sok	San	18
A	Intro MongoDB	Sao	Ry	17
B	Intro MySQL	Chan	Tina	19
B	Intro MySQL	Tith	Sophea	16

- Now it is in 1NF.
  However, it might still violate 2NF and so on.

Functional Dependencies

We say an attribute, B, has a *functional dependency* on another attribute, A, if for any two records, which have
the same value for A, then the values for B in these two records must be the same. We illustrate this as:

A → B *(read as: A determines B or B depends on A)*

employee name	project	email address
Sok San	POS Mart Sys	soksan@yahoo.com
Sao Ry	Univ Mgt Sys	sao@yahoo.com
Sok San	Web Redesign	soksan@yahoo.com
Chan Sokna	POS Mart Sys	chan@gmail.com
Sao Ry	DB Design	sao@yahoo.com

employee name → email address

Functional Dependencies (cont.)

EmpNum	EmpEmail	EmpFname	EmpLname
123	jdoe@abc.com	John	Doe
456	psmith@abc.com	Peter	Smith
555	alee1@abc.com	Alan	Lee
633	pdoe@abc.com	Peter	Doe
787	alee2@abc.com	Alan	Lee

If EmpNum is the PK then the FDs:

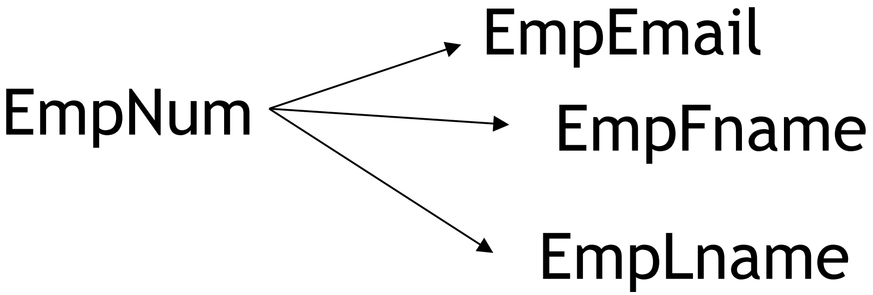
EmpNum → EmpEmail, EmpFname, EmpLname

must exist.

Functional Dependencies (cont.)

EmpNum → EmpEmail, EmpFname, EmpLname

3 different ways you might see FDs depicted

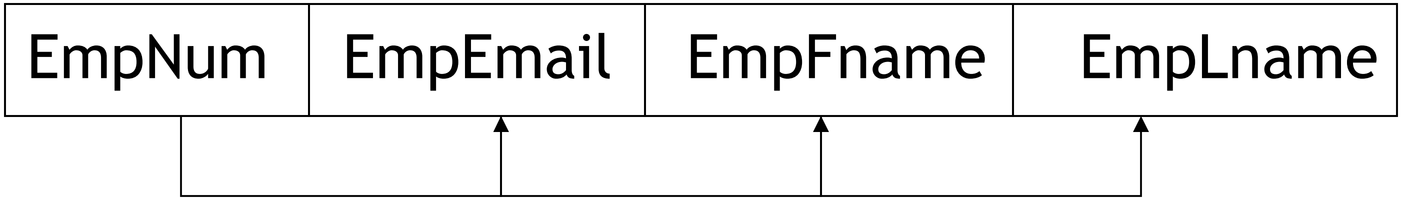


EmpNum

EmpEmail

EmpFname

EmpLname



EmpNum	EmpEmail	EmpFname	EmpLname

Determinant

Functional Dependency

EmpNum → EmpEmail

Attribute on the left hand side is known as the ***determinant***

- EmpNum is a ***determinant*** of EmpEmail

Second Normal Form (2NF)

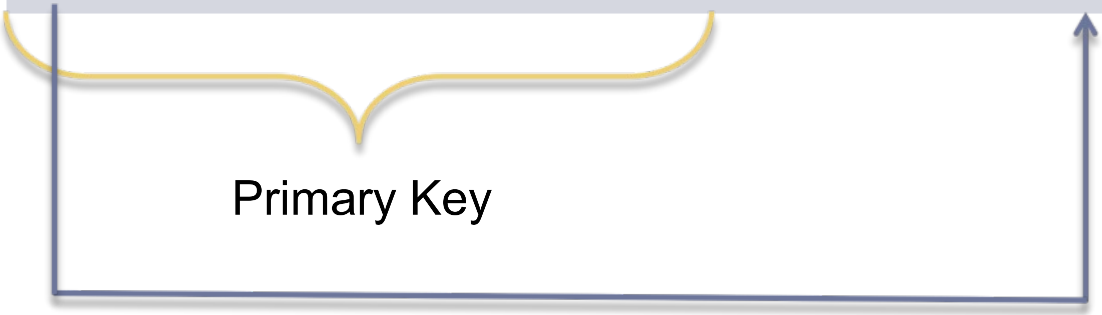
The official qualifications for 2NF are:

1. A table is already in 1NF.
2. All nonkey attributes are fully dependent on the primary key.

   *All **partial** dependencies are removed to place in another table.*

Example of a table not in 2NF:

CourseID	SemesterID	Num Student	Course Name
IT101	201301	25	Database
IT101	201302	25	Database
IT102	201301	30	Web Prog
IT102	201302	35	Web Prog
IT103	201401	20	Networking



Primary Key

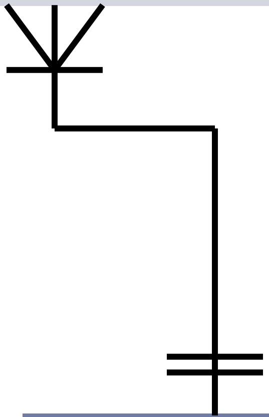
The *Course Name* depends on only *CourseID*, a part of the primary key not the whole primary {*CourseID*, *SemesterID*}.It’s called **partial dependency**.

Solution:

*Remove **CourseID** and **Course Name** together to create a new table.*

CourseID	Course Name
IT101	Database
IT101	Database
IT102	Web Prog
IT102	Web Prog
IT103	Networking

CourseID	SemesterID	Num Student
IT101	201301	25
IT101	201302	25
IT102	201301	30
IT102	201302	35
IT103	201401	20



Done? Oh no, it is still not in 1NF yet.
Remove the repeating groups too.
Finally, connect the relationship.

CourseID	Course Name
IT101	Database
IT102	Web Prog
IT103	Networking

Third Normal Form (3NF)

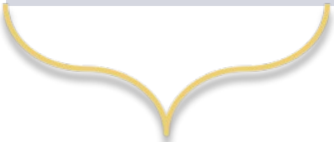
The official qualifications for 3NF are:

1. A table is already in 2NF.
2. Nonprimary key attributes do not depend on other nonprimary key attributes
   (i.e. no transitive dependencies)

   All transitive dependencies are removed to place in another table.

Example of a Table not in 3NF:

StudyID	Course Name	Teacher Name	Teacher Tel
1	Database	Sok Piseth	012 123 456
2	Database	Sao Kanha	0977 322 111
3	Web Prog	Chan Veasna	012 412 333
4	Web Prog	Chan Veasna	012 412 333
5	Networking	Pou Sambath	077 545 221



Primary Key

The Teacher Tel is a nonkey attribute, and the Teacher Name is also a nonkey atttribute. But Teacher Tel depends on Teacher Name. It is called **transitive dependency**.

Solution:

*Remove **Teacher Name** and **Teacher Tel** together to create a new table.*

Teacher Name	Teacher Tel
Sok Piseth	012 123 456
Sao Kanha	0977 322 111
Chan Veasna	012 412 333
Chan Veasna	012 412 333
Pou Sambath	077 545 221

Done?
Oh no, it is still not in 1NF yet.
Remove Repeating row.

StudyID	Course Name	T.ID
1	Database	T1
2	Database	T2
3	Web Prog	T3
4	Web Prog	T3
5	Networking	T4

Teacher Name	Teacher Tel
Sok Piseth	012 123 456
Sao Kanha	0977 322 111
Chan Veasna	012 412 333
Pou Sambath	077 545 221

ID	Teacher Name	Teacher Tel
T1	Sok Piseth	012 123 456
T2	Sao Kanha	0977 322 111
T3	Chan Veasna	012 412 333
T4	Pou Sambath	077 545 221

Note about primary key:
-In theory, you can choose Teacher Name to be a primary key.
-But in practice, you should add Teacher ID as the primary key.

Boyce Codd Normal Form (BCNF) – 3.5NF

The official qualifications for BCNF are:

1. A table is already in 3NF.
2. All determinants must be superkeys.

All determinants that are not superkeys are removed to place in another table.

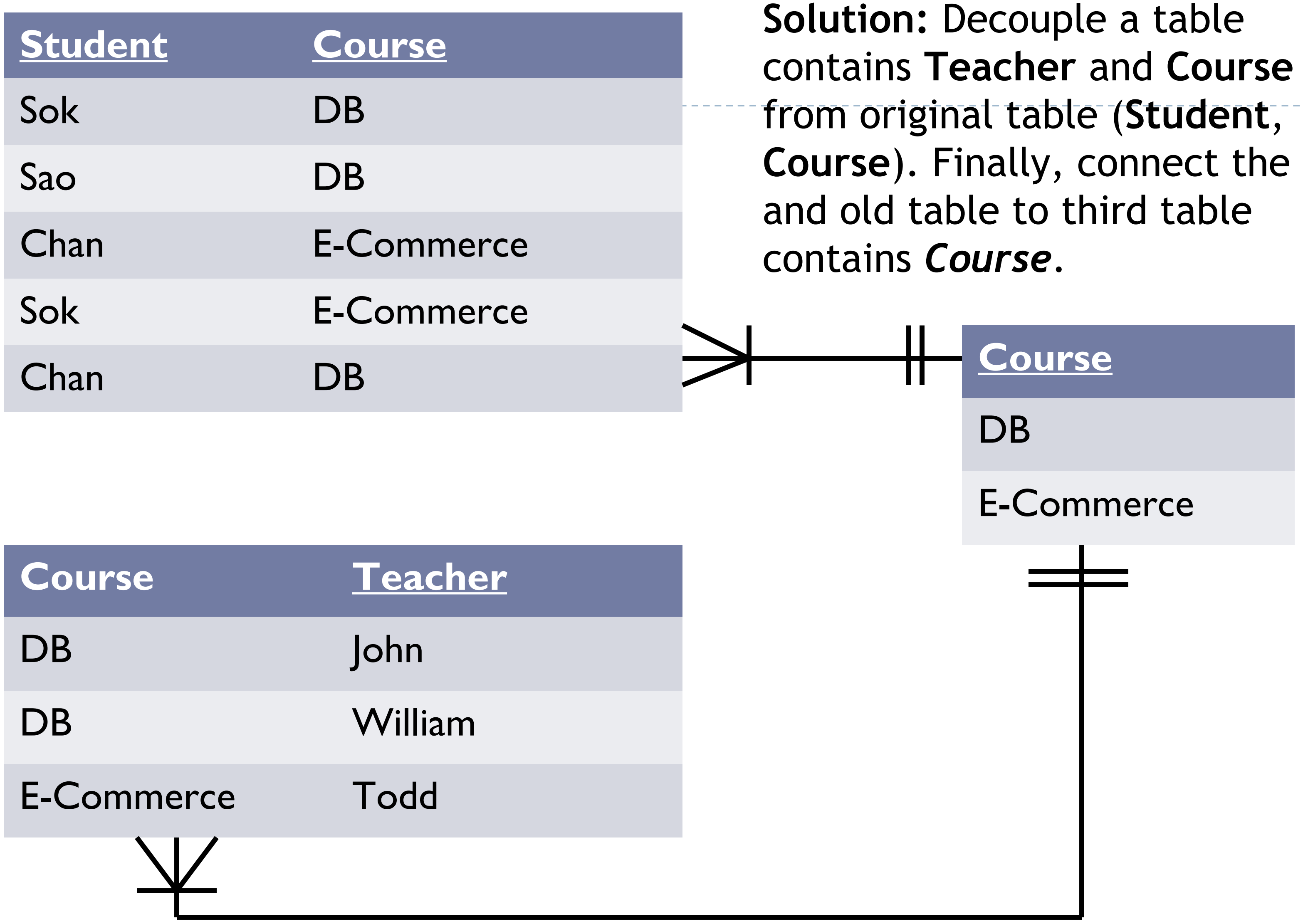
Boyce Codd Normal Form (BCNF) (Cont.)

- Example of a table not in BCNF:

Student	Course	Teacher
Sok	DB	John
Sao	DB	William
Chan	E-Commerce	Todd
Sok	E-Commerce	Todd
Chan	DB	William

- Key: {Student, Course}
- Functional Dependency:
  - {Student, Course}→ Teacher
  - Teacher → Course
- Problem: ***Teacher*** is not a superkey but determines ***Course***.

Solution: Decouple a table contains **Teacher** and **Course** from original table (**Student**, **Course**). Finally, connect the new and old table to third table contains ***Course***.



Student	Course
Sok	DB
Sao	DB
Chan	E-Commerce
Sok	E-Commerce
Chan	DB

Course
DB
E-Commerce

Course	Teacher
DB	John
DB	William
E-Commerce	Todd

DML & DDL

DML

- DML is abbreviation of **Data Manipulation Language**. It is used to retrieve, store, modify, delete, insert and update data in database.
- Examples: SELECT, UPDATE, INSERT statements

Usage & Commands

- SELECT - retrieve data from a database
- Syntax
- SELECT *column_name*,*column_name*
  FROM *table_name*;
- INSERT - insert data into a table
- Syntax
- INSERT INTO table_name
  VALUES (value1,value2,value3,etc...);
- UPDATE - updates existing data within a table
- Syntax
- UPDATE table_name
  SET column1=value1,column2=value2,...
  WHERE some_column=some_value;
- DELETE - deletes all records from a table, the space for the records remain
- Syntax
- DELETE FROM *table_name*
  WHERE *some_column=some_value*;

DDL

- DDL is abbreviation of **Data Definition Language**. It is used to create and modify the structure of database objects in database.
- Examples: CREATE, ALTER, DROP statements

Usage & commands

- CREATE - to create objects in the database.
- Syntax

-
  ```
  CREATE TABLE table_name
  (
  column_name1 data_type(size),
  column_name2 data_type(size),
  column_name3 data_type(size),
  etc...
  );
  ```
- ALTER - alters the structure of the database
- Syntax
- ALTER TABLE table_name ADD column_name datatype;
- DROP - delete objects from the database
- Syntax
- DROP TABLE table_name;
- TRUNCATE - remove all records from a table, including all spaces allocated for the records are removed
- Syntax
- TRUNCATE TABLE table_name;
- COMMENT - add comments to the data dictionary
- Syntax
- -- text_of_comment
- RENAME - rename an object
- Syntax
- RENAME TABLE old table name TO new table name;

MySQL™ The world's most popular open source database

MySQL Powers the Web

Global Top 20 Sites: Powered by MySQL

1. Google
2. Facebook
3. YouTube
4. Baidu
5. Yahoo!
6. Amazon
7. Wikipedia
8. QQ
9. Google.co.in
10. Twitter
11. **Live.com**
12. Taobao
13. **Msn.com**
14. Yahoo.co.jp
15. Sina
16. Linkedin.com
17. Google.co.jp
18. Weibo
19. **Bing.com**
20. Yandaz.ru

Source: Wikipedia 2016

They Scale with **MySQL**

Mobile Network Supporting Over **800 Million** Subscribers

2 Billion Events/Day for **Booking.com**

IDs Processed for **1 Billion** Citizens

2+ Billion Active Users

100 TB of User Data for **PayPal**

850 Million Candy Crush Game Plays/Day

MySQL Powers Social

facebook
Linked in
You Tube
WeChat
微信国际营销平台
Pinterest

MySQL Powers eCommerce

UBER
Booking.com
NETFLIX
airbnb
Alibaba.com
淘宝网
Taobao.com
Flipkart

MySQL™ Powers SaaS

New Relic®
workday®
AppDynamics
ZENEFITS
ORACLE®
RIGHTNOW CX
CLOUD SERVICE
zendesk
freshdesk
servicenow
SupplyDynamics™
AN ONI COMPANY
HubSpot
GitHub
intuit
mint

MySQL Powers FinTech

worldremit®

Square

stripe

Betterment

wealthfront

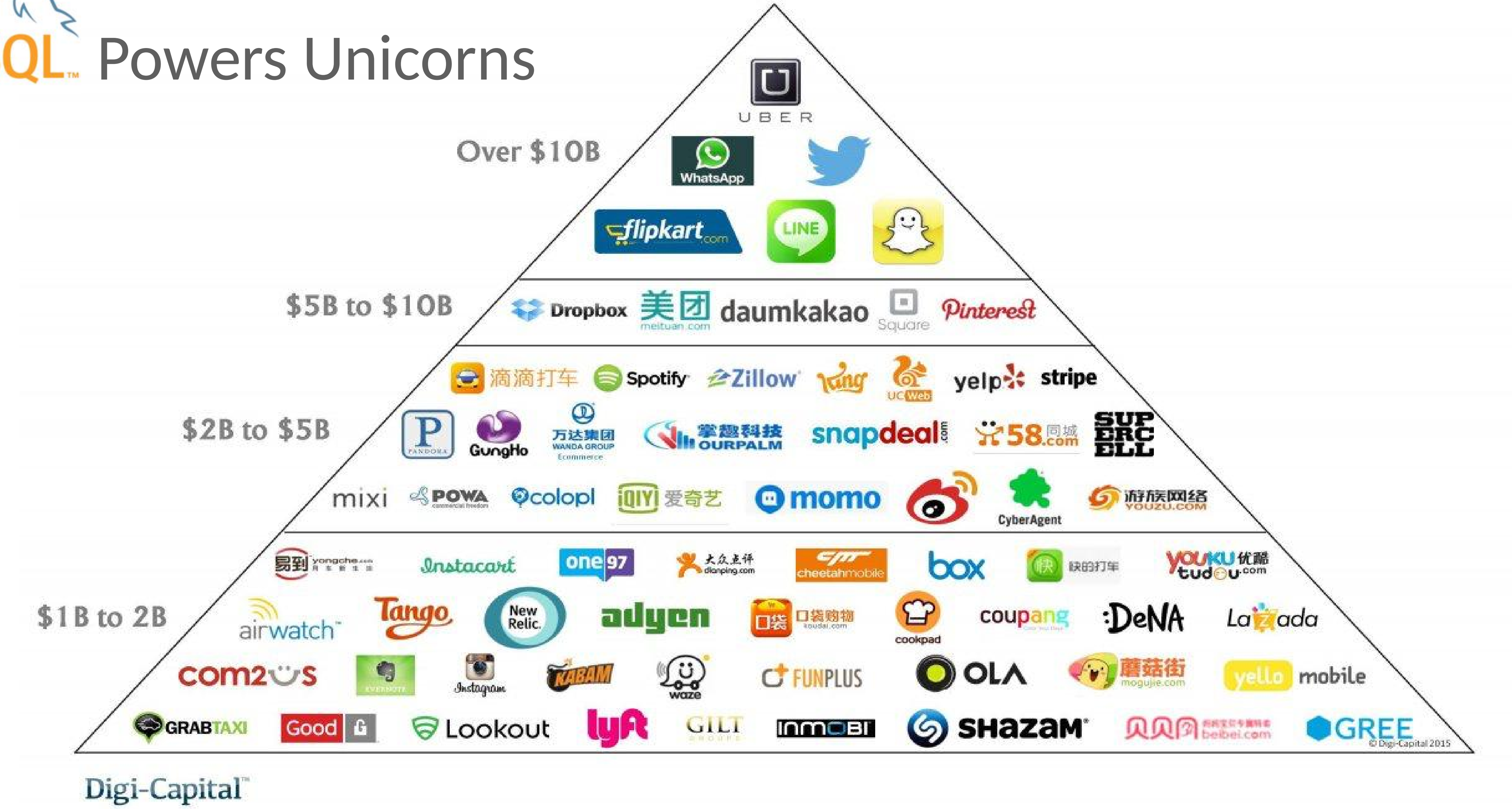
TransferWise

FutureAdvisor

LendingClub

OnDeck

MySQL Powers Unicorns



PHP PDO

Summary

	PDO	*MySQLi*
Database support	12 different drivers	MySQL only
API	OOP	OOP + procedural
Connection	Easy	Easy
Named parameters	Yes	No
Object mapping	Yes	Yes
Prepared statements (client side)	Yes	No
Performance	Fast	Fast
Stored procedures	Yes	Yes

Connection

It's a cinch to connect to a database with both of these:

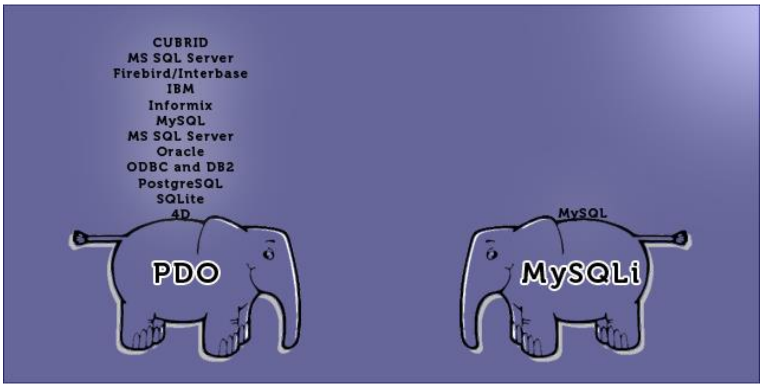
```
// PDO
$pdo = new PDO("mysql:host=localhost;dbname=database", 'username', 'password');

// mysqli, procedural way
$mysqli = mysqli_connect('localhost','username','password','database');

// mysqli, object oriented way
$mysqli = new mysqli('localhost','username','password','database');

```

Please note that these connection objects / resources will be considered to exist through the rest of this tutorial.

API Support

Both PDO and MySQLi offer an object-oriented API, but MySQLi also offers a procedural API - which makes it easier for newcomers to understand. If you are familiar with the native PHP MySQL driver, you will find migration to the procedural MySQLi interface much easier. On the other hand, once you master PDO, you can use it with any database you desire!

Database Support



The core advantage of PDO over MySQLi is in its database driver support. At the time of this writing, **PDO supports 12 different drivers**, opposed to MySQLi, which supports **MySQL only**.

To print a list of all the drivers that PDO currently supports, use the following code:

```
var_dump(PDO::getAvailableDrivers());
```

What does this mean? Well, in situations when you have to switch your project to use another database, PDO makes the process transparent. So, *all you'll have to do* is change the connection string and a few queries - if they use any methods which aren't supported by your new database. With MySQLi, you will need to *rewrite every chunk of code* - queries included.

Named Parameters

This is another important feature that PDO has; binding parameters is **considerably easier** than using the numeric binding:

```
$params = array(':username' => 'test', ':email' => $mail, ':last_login' => time() - 3600);

$pdo->prepare('
    SELECT * FROM users
    WHERE username = :username
    AND email = :email
    AND last_login > :last_login');

$pdo->execute($params);
```

...opposed to the MySQLi way:

```
$query = $mysqli->prepare('
    SELECT * FROM users
    WHERE username = ?
    AND email = ?
    AND last_login > ?');

$query->bind_param('sss', 'test', $mail, time() - 3600);
$query->execute();
```

The question mark parameter binding might seem shorter, but it isn't nearly as flexible as named parameters, due to the fact that the developer must always *keep track of the parameter order*; it feels "hacky" in some circumstances.

Unfortunately, **MySQLi doesn't support named parameters**.

Object Mapping

Both PDO and MySQLi can map results to objects. This comes in handy if you don't want to use a custom database abstraction layer, but still want ORM-like behavior. Let's imagine that we have a `User` class with some properties, which match field names from a database.

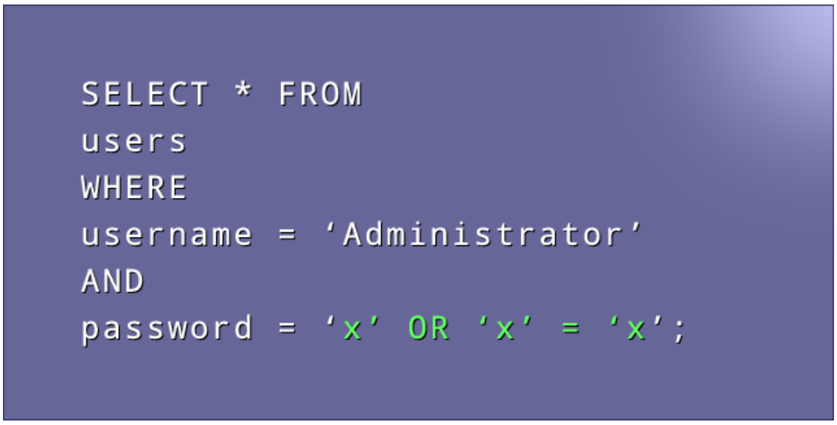
```
class User {
    public $id;
    public $first_name;
    public $last_name;

    public function info()
    {
        return '#'.$this->id.': '.$this->first_name.' '.$this->last_name;
    }
}
```

Without object mapping, we would need to fill each field's value (either manually or through the constructor) before we can use the `info()` method correctly.

This allows us to predefine these properties before the object is even constructed! For instance:

```
$query = "SELECT id, first_name, last_name FROM users";

// PDO
$result = $pdo->query($query);
$result->setFetchMode(PDO::FETCH_CLASS, 'User');

while ($user = $result->fetch()) {
    echo $user->info()."\n";
}
// MySQLI, procedural way
if ($result = mysqli_query($mysqli, $query)) {
    while ($user = mysqli_fetch_object($result, 'User')) {
        echo $user->info()."\n";
    }
}
// MySQLi, object oriented way
if ($result = $mysqli->query($query)) {
    while ($user = $result->fetch_object('User')) {
        echo $user->info()."\n";
    }
}
```

Security



```
SELECT * FROM
users
WHERE
username = ‘Administrator’
AND
password = ‘x’ OR ‘x’ = ‘x’;
```

Both libraries provide **SQL injection security, as long as the developer uses them the way they were intended** (read: escaping / parameter binding with prepared statements).

Let’s say a hacker is trying to inject some malicious SQL through the 'username' HTTP query parameter (GET):

```
$_GET['username'] = "'; DELETE FROM users; /*"
```

If we fail to escape this, it will be included in the query "as is" - deleting all rows from the `users` table (both PDO and mysqli support multiple queries).

```
// PDO, "manual" escaping
$username = PDO::quote($_GET['username']);

$pdo->query("SELECT * FROM users WHERE username = $username");

// mysqli, "manual" escaping
$username = mysqli_real_escape_string($_GET['username']);

$mysqli->query("SELECT * FROM users WHERE username = '$username'");
```

As you can see, `PDO::quote()` **not only escapes the string, but it also quotes it.** On the other side, `mysqli_real_escape_string()` will only escape the string; you will need to apply the quotes manually.

```
// PDO, prepared statement
$pdo->prepare('SELECT * FROM users WHERE username = :username');
$pdo->execute(array(':username' => $_GET['username']));

// mysqli, prepared statements
$query = $mysqli->prepare('SELECT * FROM users WHERE username = ?');
$query->bind_param('s', $_GET['username']);
$query->execute();
```

I recommend that you always use prepared statements with bound queries instead of PDO`::quote()` and `mysqli_real_escape_string()`.

Performance

While both PDO and MySQLi are quite fast, MySQLi performs insignificantly faster in benchmarks - ~2.5% for non-prepared statements, and ~6.5% for prepared ones. Still, the native MySQL extension is even faster than both of these. So, if you truly need to squeeze every last bit of performance, that is one thing you might consider.

Summary

Ultimately, PDO wins this battle with ease. With support for twelve different database drivers (eighteen different databases!) and named parameters, we can ignore the small performance loss, and get used to its API. From a security standpoint, both of them are safe as long as the developer uses them the way they are supposed to be used (read: prepared statements).

So, if you're still working with MySQLi, maybe it's time for a change!