ORM - Object-relational mapping

Ján Balažia & Jakub Dubec

O čom to bude?

1. RDBMS a ORM

Problémy spojené s RDBMS a základné riešenia cez ORM v kocke

2. Návrhové vzory

Kopec buzzword-ov ako Active record, Data mapper, STI, TPC, TPCC, Lazy loading...

3. Migrácie

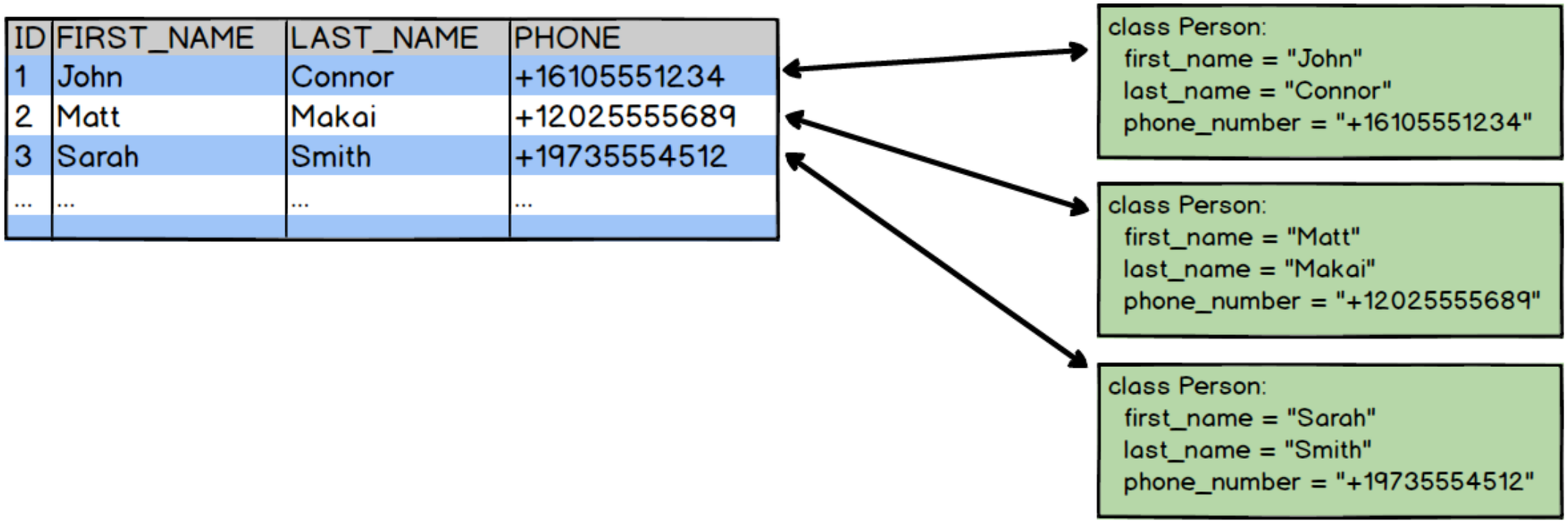
Ako sa nezblázniť pri kolaborácii a deployoch

4. Live programovanie

1. RDBMS a ORM

RDBMS a ORM

ORM je most medzi tabuľkami, ich vzťahmi a stĺpcami preložený do objektov



V prístupe bez ORM dostanem Tuple, v lepšom prípade Dictinary

ORM a databázový server

V ideálnom svete je jedno API a je nezávislé od databázového serveru - app logika je nezávislá

napr. Django ORM podporuje PostgreSQL, MariaDB, MySQL, Oracle, SQLite

Ošetrovanie dátových typov pre každú databázu

MySQL tinyint(1) vs. boolean v Postgres

Časové zóny Postgress má a MariaDB stále nič

MariaDB sa spraví automatický preklad na UTC resp. podľa nastavenia

Data Definition Language (DDL)

CREATE, ALTER, DROP, RENAME, ...

```
class Posts(models.Model):
    class Meta:
        db_table = “posts”

    author = models.CharField(max_length=256, null=False)
    title = models.CharField(max_length=256, null=False)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now_add=True)
    deleted_at = models.DateTimeField(null=True, blank=True)
```

Zmeny reflektované cez migrácie

Data Query Language (DQL)

```
SELECT ... FROM ... WHERE ...
```

```
posts = Post.objects.filter(author="Jan")
```

Transaction Control Language (TCL)

```
BEGIN, COMMIT, ROLLBACK, SAVEPOINT
```

```
with transaction.atomic():
    your_atomic_block()
```

Data manipulation language (DML)

INSERT INTO ... VALUES ...

UPDATE ... SET ... WHERE ...

DELETE FROM ... WHERE ...

```
post = Post.objects.create(author="Jan", title='Test')
```

```
post.author = "Jakub"
```

```
post.save()
```

```
post.delete()
```

Prečo ORM?

Udržateľnosť a efektivita

neopakuje sa kokód dokola, jednoducho čitatelné - neukecané a priamočiare, migrácie pri zmenách štruktúry, exekúcia len keď treba

Flexibilita voči databázovej vrstve

Objektový prístup ako vo zvyšku aplikácie - zachovanie paradigmy

Bezpečnosť

Surový vstup od používateľa nejde priamo k DB

Bezpečnosť

```
SELECT password FROM users WHERE
username = 'sudruh.blaha' LIMIT 1;
```

```
SELECT password FROM users WHERE
username = 'sudruh.blaha' OR 1=1; --' LIMIT 1;
```

Ako?

Prepared statements a čistenie dát

```
PREPARE users(varchar(64)) AS
SELECT password FROM users WHERE
username=$1 LIMIT 1;
EXECUTE users('sudruh.blaha" OR 1=1; --');
```

2. Návrhové vzory

Active record vs. Data mapper

Active record

klasické CRUD aplikácie

Ruby on Rails, Laravel, Django ORM

Priame mapovanie databázy do objektov spojené s logikou - new, set property, save

Data mapper

striktné pravidlá a procedúry

Vrstva zodpovedná za transfer dát medzi databázou a pamäťou. Napr. SQLAlchemy či Doctrine2.

Vytvorenie spojenia, vytvorenie záznamu, pridanie záznamu do spojenia, komitnutie spojenia.

Table Data Gateway, Row Data Gateway

Row Data Gateway

Manažér pre prácu s konkrétnym záznamom

Definícia štruktúry, metód a vlastností, ktoré reprezentujú konkrétny riadok v databáze (User.updateName())

Table Data Gateway

Manager pre prácu s tabuľkami

Definícia metód napríklad pre find, update insert, delete aplikovaných pre konkrétnu štruktúru reprezentujúcu model (Users.findByName())

STI, TPC, TPCC

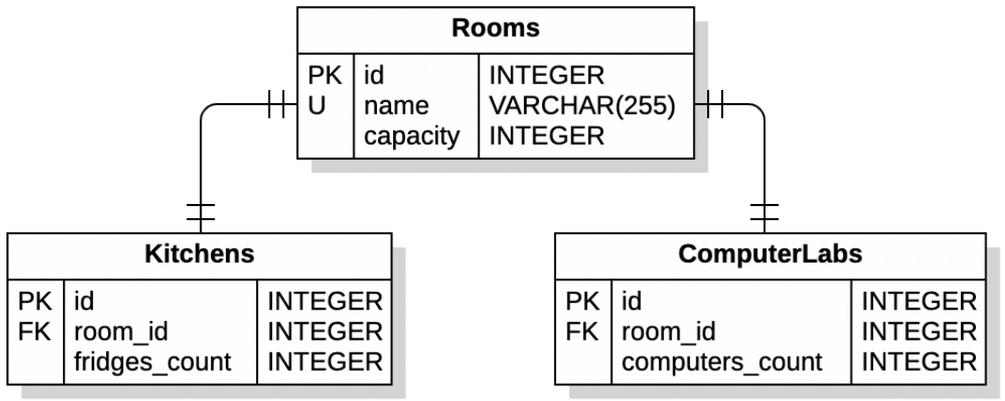
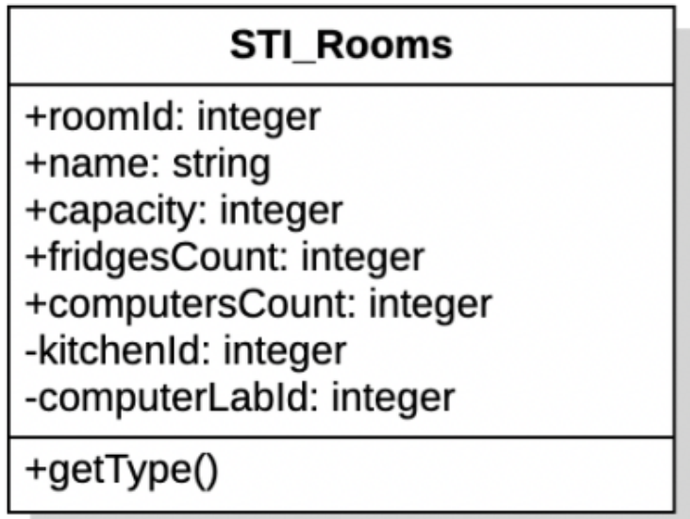


Table per Class Inheritance (TPC)
Table per Concrete Class Inheritance (TPCC)



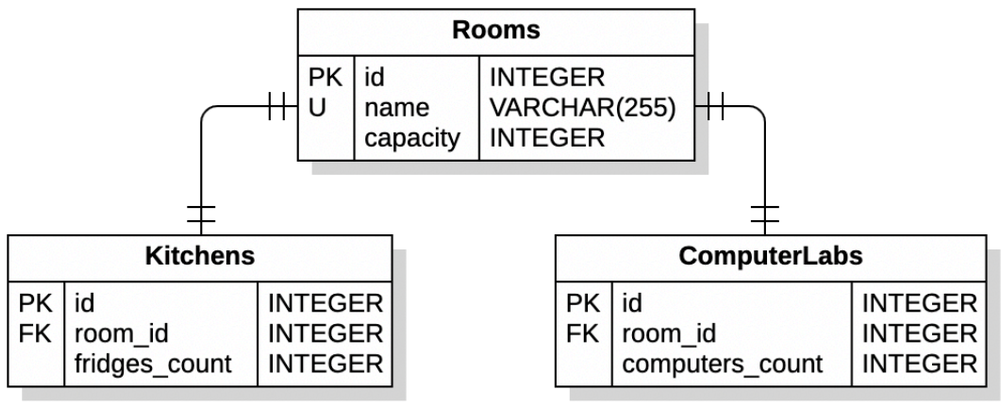
Single Table Inheritance (STI)

Spojené tabuľky mapuje do jedného objektu

Obsahuje agregované informácie o type záznamu, ktorý uchováva

Vhodný len na tabuľky, ktoré zdieľajú spoločné dáta

STI, TPC, TPCC



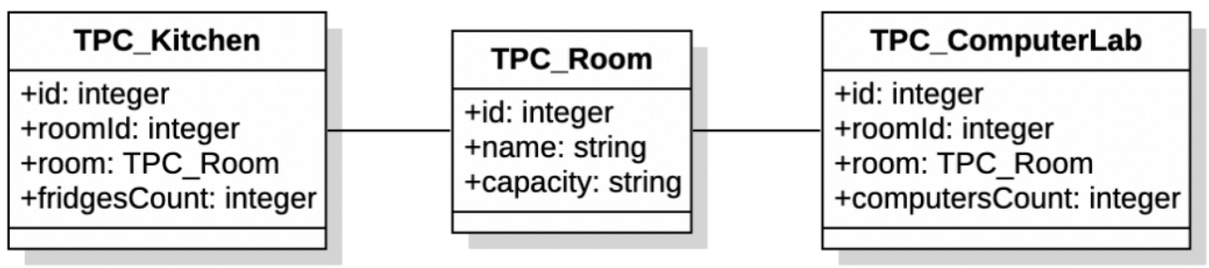


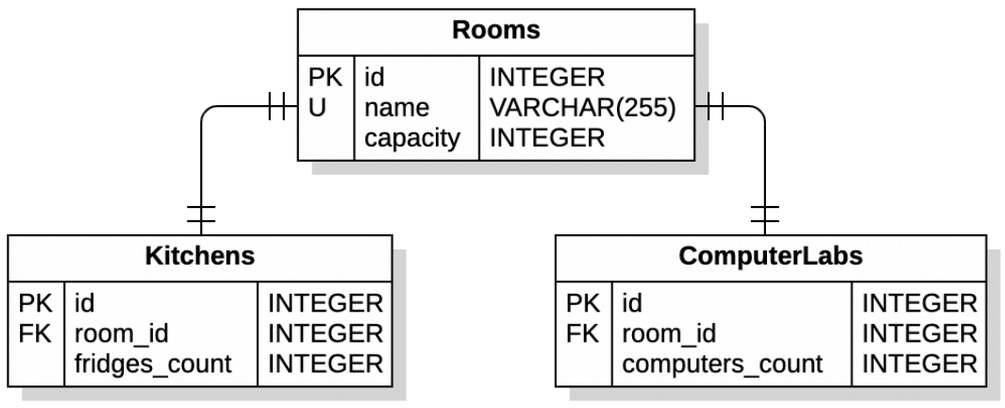
Table per Class Inheritance (TPC)

Pre každú mapovanú tabuľku jeden objekt

Vzťahy, medzi tabuľkami budú mapované pomocou asociácie / kompozície

Najštandardnejšie používané

STI, TPC, TPCC



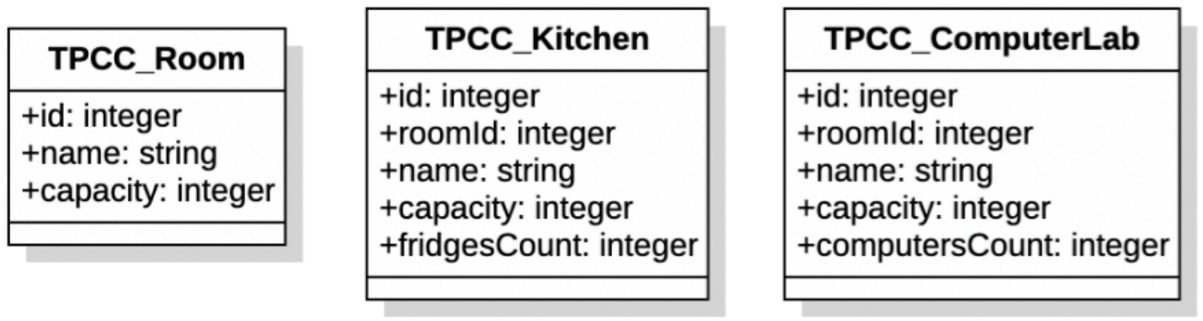


Table per Concrete Class Inheritance (TPCC)

Kombinácia STI a TPC

Potomok v hierarchii preberie aj všetky atribúty - nastáva teda redundancia dát

Využívané z aplikačných dôvodov (napr. serializácia)

Lazy loading, Intentity map

Lazy loading

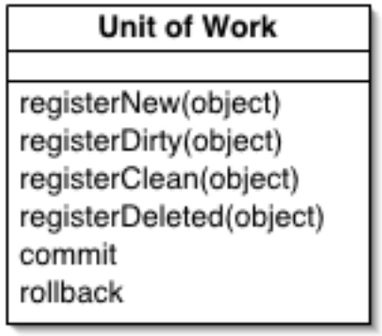
Vykonávam query až keď naozaj potrebujem dáta

Napr. postupné vyskladávanie podmienok, či iterovanie s aplikačnou logikou (len v iterácii, ktorá potrebuje dáta sa vykoná query)

Identity map

Ak už mám výsledok v pamäti, tak sa query nevykoná

Unit of work



Biznis logika transakcie

Jednotlivé hodnoty záznamu v ORM počas transakcie môžu byť v stave New, Dirty, Clean, Deleted

Komitujú sa až výsledné hodnoty po dokončení biznis logiky procesu

3. Migrácie

Čo sú to migrácie a prečo sú dôležité?

Migrácie sú postupnosti zmien štruktúr databázy od jej vytvorenia až po aktuálny stav

Prečo? Lebo nikto neľúbi, keď niekto na neho vrieska, prečo nič nefunguje

Nová aplikačná logika často vyžaduje nové modely, či úpravu existujúcich

Úpravy nad modelmi == databázové úpravy

Migrácia dát zo štruktúry A na B

Potreba synchronizovať deploy aplikácie a databázové úpravy

Ako fungujú?

Každý framework má vlastný štýl, ako si zaznamenáva vykonané zmeny v DB

Princíp: výsledný stav všetkých doteraz sekvenčne aplikovaných zmien porovná s aktuálnymi modelmi

Z rozdielu vytvorí novú migráciu, ktorej dá zavislosť na poslednej vykonanej

Aplikácia vďaka aplikovaní migrácii vie, či má správne pripravenú databázu pre beh

Preto sa databázové zmeny nerobia manuálne

4. Ukážka.