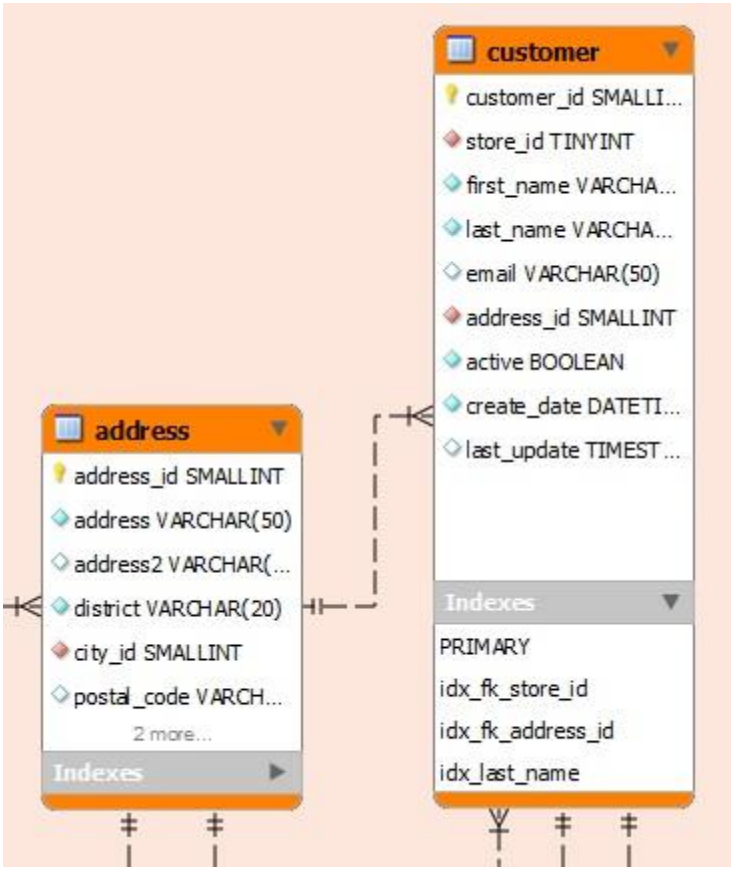
Povezivanje podataka više tabela

Sve do sada, svaki upit sa SELECT naredbom koji smo pisali dobavljao je podatke samo iz jedne tabele. Ipak, u relacionim bazama podataka, mi uglavnom baratamo sa različitim brojem povezanih tabela. Zbog toga ćemo često doći u situaciju da nam je potrebna primena SELECT naredbe u cilju kombinovanja podataka više kolona.

U abecedi SQL upita, spajanje tabela je slovo B. Nakon poznavanja osnove upitne sintakse, ovo je svakako sledeća oblast koju je potrebno preći.

Studija slučaja

Uzmimo u razmatranje dve tabele koje postoje u okviru Sakila baze podataka. To su tabele customer i address.



Slika – Dijagram tabela customer i address

Ono što možemo da primetimo je da u okviru tabele *customer* ne postoje konkretne informacije o na primer imenu ulice mušterije, već su takvi podaci grupisani u zasebnoj tabeli *address*. Još jedna stvar koju primećujemo je da obe prikazane tabele poseduju *atribut address_id*. O ovome sam već govorio u prošlim lekcijama, kada sam objašnjavao

relacioni model, i može se zaključiti da je atribut *address_id* u okviru tabele customer zapravo strani ključ, čiji je cilj povezivanje ove dve tabele.

Logično je sada postaviti pitanje kako da dobijemo prikazane mušterije sa ulicom u kojom žive. SELECT upitom nad jednom tabelom to nećemo postići.

Prvi način koji bismo mogli da upotrebimo je da napišemo nešto ovako:

```
SELECT first_name, last_name, address FROM customer, address;
```

Ukoliko pažljivo analiziramo rezultate upita koje smo dobili možemo da primetimo da je upit koji smo definisali vratio sve moguće kombinacije imena i prezimena, i adrese. U velikim bazama podataka ovo bi izazvalo pravu katastrofu, a uz to ne bi vratilo željeni rezultat.

Jedan način da ovaj upit popravimo kako bi vratio željeni rezultat je da navedemo uslov pojavljivanja rezultata, tj. da navedemo relaciju između tabela. To možemo postići korišćenjem ključne reči WHERE sa kojom smo se već upoznali:

```
SELECT first_name, last_name, address
FROM customer, address
WHERE address.address_id = customer.address_id;
```

Na ovaj način dobićemo rezultat koji očekujemo, ali je samo rešenje vrlo neefikasno i sporo, tako da se za spajanje tabela koriste takozvani JOIN-ovi.

JOIN

Join je ključna reč u SQL-u, koja označava spoj između dve tabele. Ova ključna reč nikada se ne pojavljuje sama, već obavezno dolazi u kompletu sa ključnom rečju ON (ali često i nekim drugim ključnim rečima).

Pogledajmo sada kako bi prethodni primer izgledao upotrebom JOIN-a.

```
SELECT customer.first_name, customer.last_name, address.address
FROM customer JOIN address
ON address.address_id = customer.address_id
```

Analizirajmo sada ovaj upit deo po deo.

Prvo smo naveli kolone koje želimo da se pojave u okviru rezultata. Primećujete da smo naveli kao prefiks i naziv tabele kojoj kolona pripada. Ovo nije obavezno raditi, ali se preporučuje iz dva razloga. Prvo, poboljšava preglednost, a drugo sprečava pojavu greške u situacijama kada kolona sa istim nazivom postoji u obe tabele.

U sledećoj liniji navedene su same tabele koje se spajaju i to tako što je između naziva tabela upotrebljena ključna reč JOIN.

U poslednjoj liniji zadat je kriterijum povezivanja. U to svrhu se koristi ključna reč ON, nakon koje je definisan uslov. Ukoliko su nazivi atributa u tabelama identični, što će najčešće i biti slučaj, i što jeste slučaj u našem primeru, obavezno je navođenje naziva tabele kao prefiksa.

Generalno, postoji, nekoliko vrsta JOIN-ova sa kojim će se upoznati:

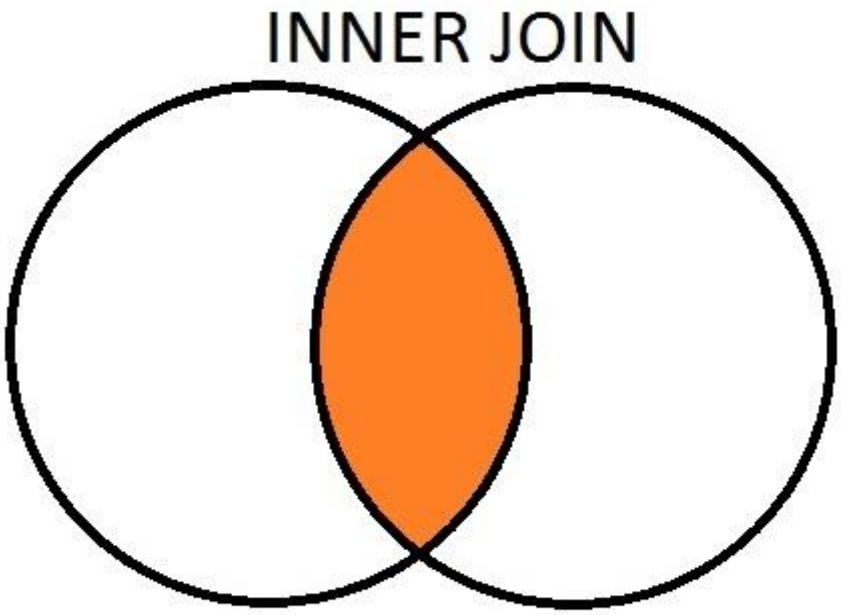
- Inner Join,
- Left Join,
- Right Join,
- Full Join.

Prilikom povezivanja podataka iz više tabela može se dogoditi da određeni zapisi u nekoj od tabela nemaju vrednosti. U našem slučaju to bi se dogodilo ukoliko korisnik ne bi imao unetu ulicu (*address*) u tabeli address, ili ako ne bi imao uneto ime ili prezime u tabeli *customer*. Boljim uvidom u ove tabele i unete podatke možemo da vidimo da svaki zapis u bazi sadrži ove podatke. Ali, možemo kao primer uzeti drugu kolonu tabele *address*, a to je*district*. Ova kolona nema vrednost za baš sve zapise i dobro će nam poslužiti za demonstraciju različitih vrsta JOIN-a.

Inner JOIN

Inner Join je zapravo Join koji smo videli na prethodnom primeru. Takoreći, potpuno identičan efekat ima korišćenje ključne reči JOIN i INNER JOIN, i u oba slučaja biće selektovani redovi koji za definisane kolone imaju vrednosti u obe tabele. Ovo znači da, ako govorimo hipotetički, ukoliko neki od naših mušterija nemaju unetu adresu, oni neće biti prikazani u rezultatima.

Grafički bi to moglo da se prikaže ovako:



Slika – Inner Join

Napišimo sada upit za prikaz svim mušterija, sa podacima o imenu, prezimenu, adresi i distriktu u kojom mušterija živi.

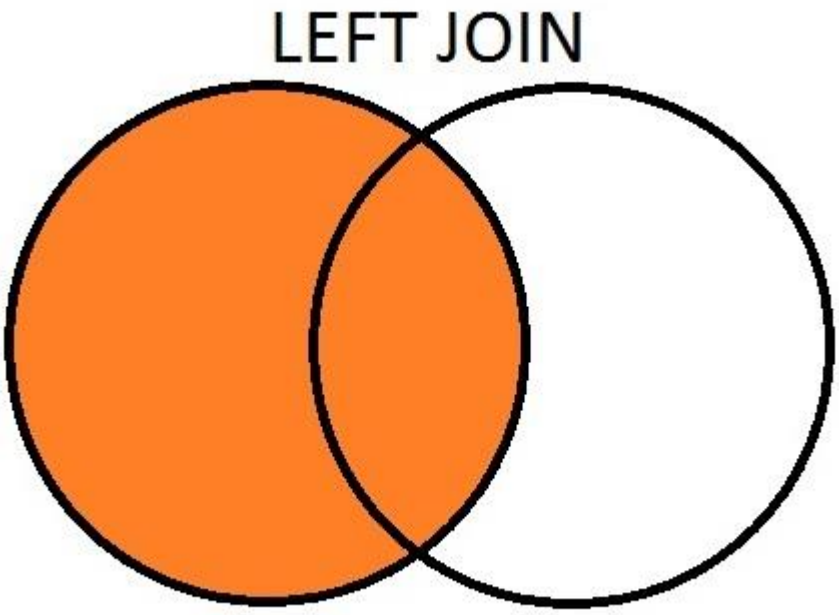
```
SELECT customer.first_name, customer.last_name, address.address,
address.district
FROM customer INNER JOIN address
ON address.address_id = customer.address_id
```

Pored podataka koje smo dobili, ukoliko koristite MySQL Workbench možete pročitati i broj pronađenih redova:

```
599 row(s) returned
```

Left JOIN

Spajanje tabela definisano LEFT JOIN-om vratiće sve zapise iz leve tabele (tabele A), pa čak i ako neki od zapisa za tražene kolone nema vrednost.



Slika – Left Join

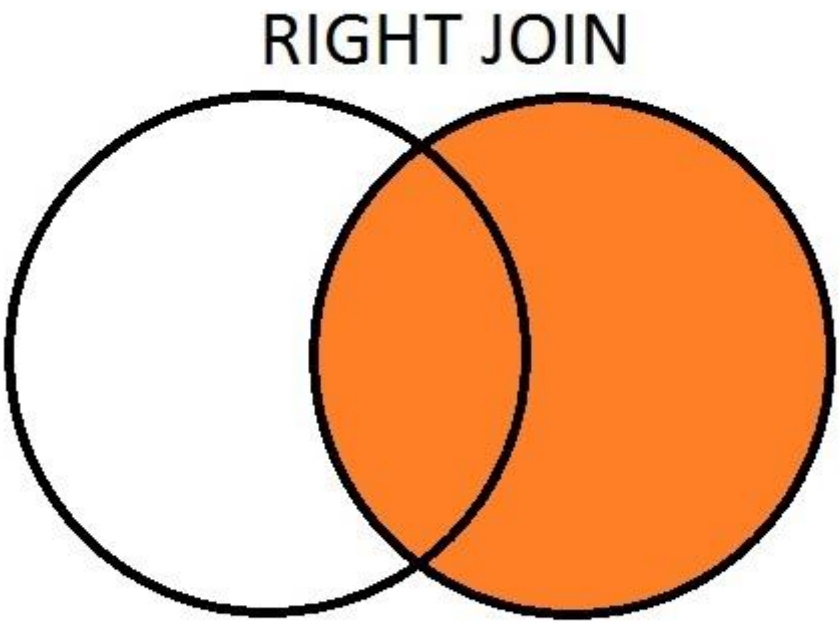
```
SELECT customer.first_name, customer.last_name, address.address,
address.district
FROM customer LEFT JOIN address
ON address.address_id = customer.address_id
```

Promenom INNER JOIN-a u LEFT JOIN, dobijamo identičan rezultat izvršenja upita, zato što svaki zapis iz tabele customer poseduje vrednosti za kolone *first_name i last_name*. TO možemo videti i uvidom u broj vraćenih redova:

```
599 row(s) returned
```

Right JOIN

Slično kao kod prethodnog JOIN-a, sa RIGHT JOIN-om dobićemo sve zapise iz desne tabele (tabele B), i samo one koji imaju vrednosti za definisane kolone iz tabele A.



12.4 – Right Join

```
SELECT customer.first_name, customer.last_name, address.address,
address.district
FROM customer RIGHT JOIN address
ON address.address_id = customer.address_id
```

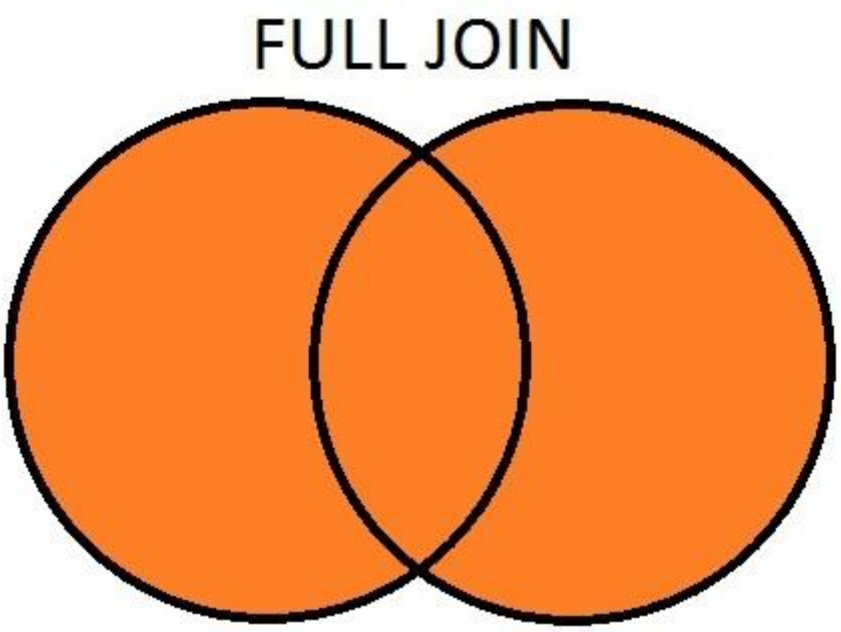
Kada izvršimo ovaj upit, možemo da primetimo da je broj vraćenih redova veći nego u prethodnim slučajevima:

```
603 row(s) returned
```

Ovo se dešava, kao što smo već rekli, zato što su sada u vraćene zapise ušli i oni koji nisu vezani spoljašnjim ključem, a koji u prethodnim upitima nisu uvrštavani u rezultate.

Full JOIN

Po SQL specifikaciji postoji nešto što se naziva FULL JOIN. Šta bi, zapravo, jedan ovakav JOIN proizveo, najbolje je videti na sledećoj slici:



12.5 – Full Join

Ovako nešto nije podržano u MySQL-u na način da postoji ključna reč kao što je to slučaj sa LEFT i RIGHT JOIN-ima. Da bi se selektovali zapisi iz obe tabele, pa čak i ako oni nemaju odgovarajuće podatke u svim navedenim kolonama, može se koristiti UNION.

UNION je zapravo ključna reč koja služi za spajanje više result set-ova, tj. za spajanje rezultata više SELECT naredbi. Mi ovu naredbu (UNION) možemo iskoristiti za simuliranje Full JOIN-a:

```
SELECT customer.first_name, customer.last_name, address.address,
address.district
FROM customer LEFT JOIN address
ON address.address_id = customer.address_id
UNION
SELECT customer.first_name, customer.last_name, address.address,
address.district
FROM customer RIGHT JOIN address
ON address.address_id = customer.address_id
```

Praktično, napisali smo dva identična upita, jedan sa LEFT JOIN-om, drugi sa RIGHT JOIN-om, između kojih smo iskoristili ključnu reč UNION.

Dobro je napomenuti da se korišćenjem naredbe UNION automatski eliminišu rezultati koji su duplikati, dok se u situacijama kada su potrebni i duplikati može koristiti UNION ALL.

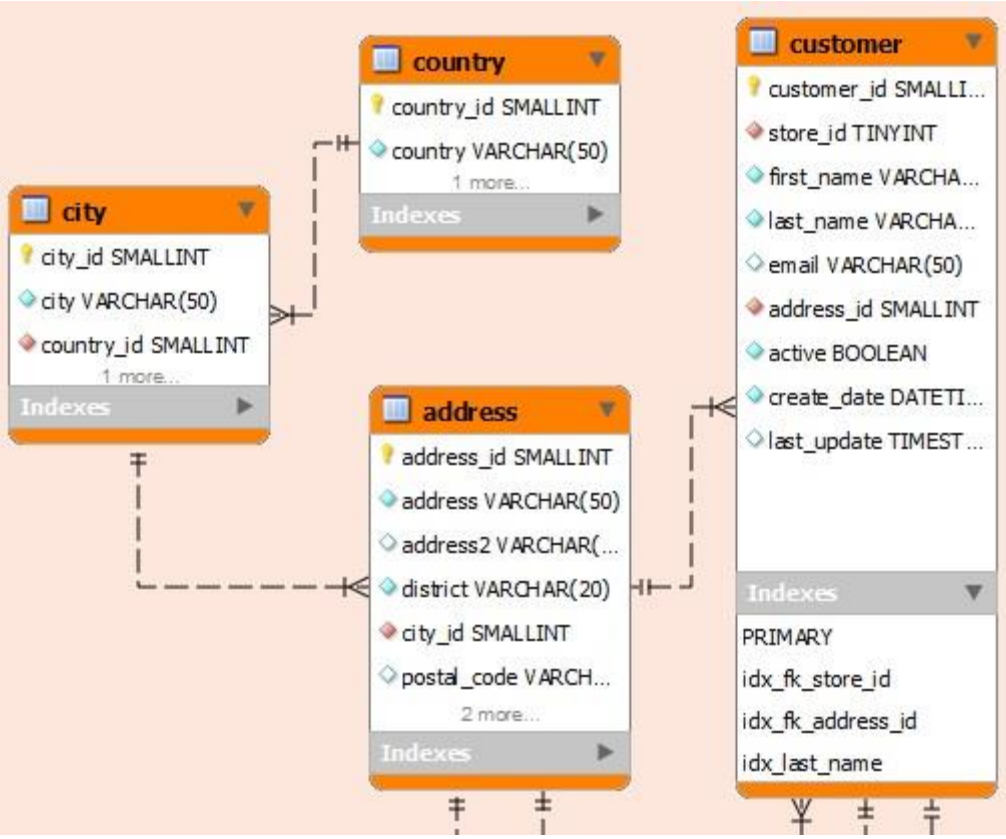
Pomenućemo još dve veoma bitne stvari kada je reč o UNION-u:

- broj kolona iz kojih se selektuju podaci pojedinačnih SELECT naredbi mora biti identičan;
- tipovi kolona koji se nalaze na istim pozicijama u SELECT upitima moraju biti identični ili barem konvertibilni (kompatibilni).

Spajanje više od dve tabele

U prethodnim primerima ove lekcije mi smo vršili spajanje samo dve tabele. Pogledajmo sada na koji način se može izvršiti spajanje više od dve tabele.

Pogledajmo tabele vezane za podatke o mušterijama koje postoje u okviru Sakila baze podataka:



12.6 – Dijagram tabela za smeštanje podataka o mušterijama

Tabele *customer* i *address* su nam poznate iz dosadašnjeg toga lekcije, a sada u igru ubacujemo još dve povezane tabele, što daje ukupno 4 povezane tabele.

Ukoliko želimo da prikažemo podatke o mušteriji, među kojima će biti i naziv države, naziv grada, adresa, ime i prezime, moraćemo da izvršimo spajanje sve četiri prikazane tabele.

```
SELECT customer.first_name, customer.last_name, address.address,
address.district, city.city, country.country
FROM customer
JOIN address ON address.address_id = customer.address_id
JOIN city ON city.city_id = address.city_id
JOIN country ON country.country_id = city.country_id
```

Vežbe

Vežba1

Problem

Potrebno je dobiti podatke o svim filmovima u bazi Sakila, kao i o jeziku svakog filma.

Rešenje

Očigledno je da je potrebno izvršiti spajanje dve tabele, i to tabele film i tabele language. Naravno, pre bilo čega je potrebno naznačiti bazu sa kojom radimo:

```
use sakila;
```

Nakon ovoga možemo napisati upit:

```
SELECT film.title, language.name
FROM film INNER JOIN language
ON film.language id = language.language id;
```

Vežba2

Problem

Potrebno je dobiti listu svih filmova sa podatkom o odgovarajućem žanru filma.

Rešenje

U okviru tabele film ne postoji podatak o kategoriji, tj. žanru filma. Žanrovi se čuvaju u zasebnoj tabeli category, dok su povezanosti, tj. relacije između tabela film i category obrađene u zasebnoj tabeli. To je tabela film_category, tako da je potrebno, zapravo, izvršiti spajanje tri tabele:

```
SELECT film.title, category.name
FROM film INNER JOIN film_category
ON film.film_id = film_category.film_id
INNER JOIN category
ON category.category_id = film_category.category_id
```