

# **Databases, ORM and Database Operation (part 2)**

## **Gold - Chapter 4 - Topic 3 (part 2)**

Selamat datang di **Chapter 4 Topic 3**  
online course **Back End Java** dari  
**Binar Academy!**

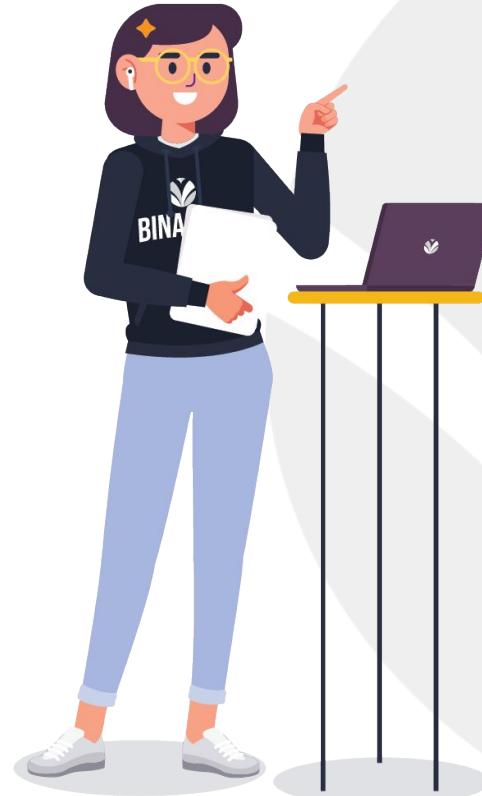


# Welcome back, Binarian! ☺

Pada part 1, kita udah belajar tentang konsep dari Relational Database dan ORM.

Lanjut di part 2 ini, kita bakal belajar tentang lebih dalam mengenai **Database Operation**. Mulai dari dasar-dasar query, aggregate query, sub query, desain database menggunakan ERD, sampai Normalization.

Yuk, langsung aja kita kepoin~



**Dari sesi ini, kita bakal bahas hal-hal berikut:**

- Pengantar dasar Query
- Syntax dari Query - Aggregate Query
- Syntax dari Query - Sub Query
- Bagaimana desain database dengan ERD
- Konsep normalization



Untuk mengawali topic kali ini, materi yang bakal kita bahas yaitu **Dasar Query**.

Kira-kira kamu udah ada bayangan tentang Query belum?

Kalau belum, kita cari tahu bareng-bareng, yuk!



### Kamu masih ingat nggak, kalau kamu udah kenalan sama query DDL dan DML di topic sebelumnya?

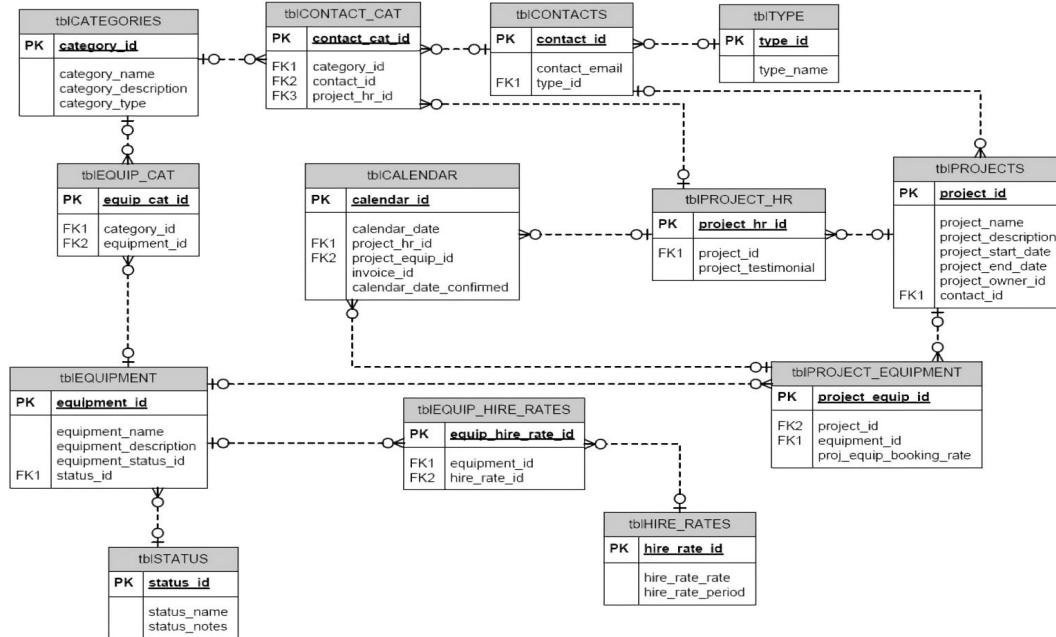
Pada topic ini, kamu bakal belajar beberapa query read lebih dalam lagi dengan fokus utamanya, yaitu **query join**, **aggregation**, dan juga **sub query**.

Nggak cuma itu, di topic ini kamu juga bakal mempelajari tentang gimana relasi table bekerja.

Pasti kamu penasaran, kan?



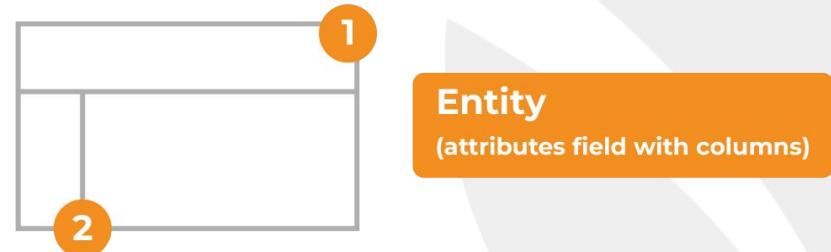
Coba perhatikan contoh dari tabel ERD pada gambar di bawah ini dulu, ya!



Dari ERD tersebut kita bisa lihat berbagai macam table, lho~

Simbol di samping merupakan suatu entity yang akan kita bedah strukturnya:

- Kolom teratas dipakai untuk menampilkan nama entity.
- Kolom yang di-split di bawahnya dipakai untuk menampilkan nama field/kolom dan keterangan apakah kolom tersebut merupakan primary key atau foreign key.



Berikut adalah beberapa syntax query dasar untuk melakukan query read pada satu table!

### 1. Select \* from projects

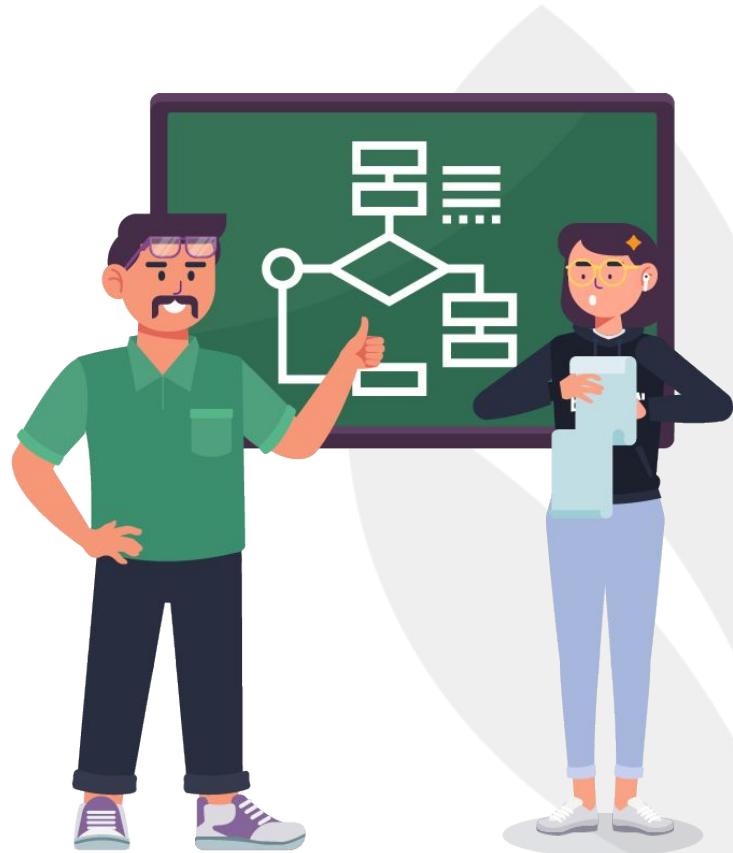
Tanda \* menandakan kalau kita mau menampilkan semua kolom pada table. Query kayak gini bakal mengambil seluruh row yang ada di dalam table projects.



### 2. Select project\_name from projects

project\_name merupakan nama dari salah satu field di table projects.

Query ini bakal menarik seluruh row yang ada di dalam table projects tapi cuma menampilkan kolom project\_name aja.



### 3. `Select p.project_name as nama_proyek from projects p`

p merupakan alias dari table. **Alias ini bisa bantu untuk mempersingkat penulisan nama table pada query yang kompleks.**

Sedangkan nama proyek merupakan alias untuk nama kolom saat penampilan data.

Query ini menghasilkan data yang sama kayak query yang sebelumnya, tapi pas ditampilkan, nama kolom dari `project_name` ini adalah nama proyek.



Untuk melakukan spesifikasi kriteria, ada dua cara Syntax Query yang bisa dipakai nih, gengs~

Syntax query ini yaitu:

1. Syntax Query untuk melakukan spesifikasi kriteria dari data yang dicari, dan
2. Syntax Query untuk mencari data yang punya data lebih dari satu kriteria.

Kita bahas lebih lanjut, ya!



### Syntax query untuk melakukan spesifikasi kriteria dari data yang dicari

Ada tiga syntax query yang bakal kita pelajari, nih:

- Query yang ini bakal menampilkan seluruh row dari table projects yang punya value 234 pada field owner\_id



```
Select * from projects where owner_id = 234
```

- Query ini bakal menampilkan seluruh row dari table projects yang punya value dengan prefix djarum pada field project\_name



```
Select * from projects where project_name like 'jarum%'
```

- Query di samping bakal menampilkan seluruh row dari table projects yang punya value dengan akhiran sempurna pada field project



```
Select * from projects where project_name like '%sempurna'
```

Syntax query untuk mencari suatu data yang punya data lebih dari satu kriteria

Berikut adalah tiga syntax query yang bisa dipakai~

- Query pertama bakal mengambil seluruh row yang punya owner\_id bernilai 234 dan juga bakal menarik row yang punya owner\_id bernilai 76.

```
Select * from projects where owner_id = 234 or owner_id=76
```

- Query yang kedua bakal mengambil seluruh row yang memiliki owner\_id bernilai 234 dan juga akan menarik row yang punya project\_name dengan akhiran akhiran mild.



```
Select * from projects where owner_id = 234 or project_name='%mild'
```

- Query yang ketiga bakal mengambil seluruh row yang punya owner\_id bernilai 234 dan project\_name dengan akhiran mild.

Beda kayak query sebelumnya, data ini bisa menghasilkan data yang lebih sedikit (nggak bakal melebihi jumlah data dari row pada query sebelumnya) karena di row yang sama, kriteria owner\_id = 234 and project\_name='%mild' harus terpenuhi.

Sedangkan di query sebelumnya owner\_id = 234 or project\_name='%mild' cuma salah satu aja yang perlu dipenuhi.

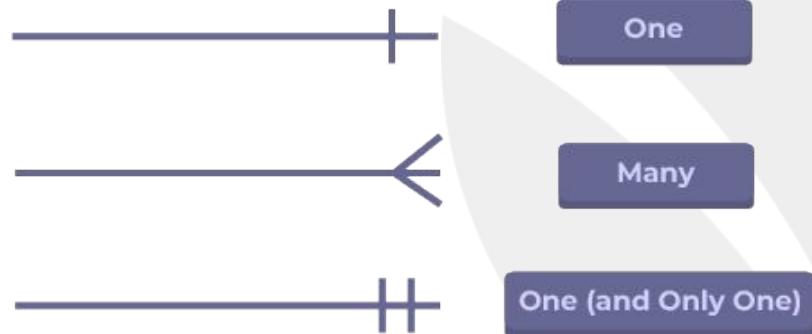


```
Select * from projects where owner_id = 234 and project_name='%mild'
```

**Perhatikan garis di samping ya, lalu simak penjelasannya di bawah ini~**

Berurutan dari yang paling atas penjelasannya akan gini:

1. Cuma punya satu row aja dan nggak bisa kalau nggak punya row sama sekali.
2. Bisa punya banyak row dan nggak bisa nggak punya row sama sekali.
3. Antara satu dengan lainnya punya satu row aja.



4. Cuma punya satu row aja dan diperbolehkan nggak punya row.
5. Punya satu row dan lainnya punya banyak row.
6. Punya lebih dari satu row dan diperbolehkan nggak punya row sama sekali.



Zero or One



One or Many

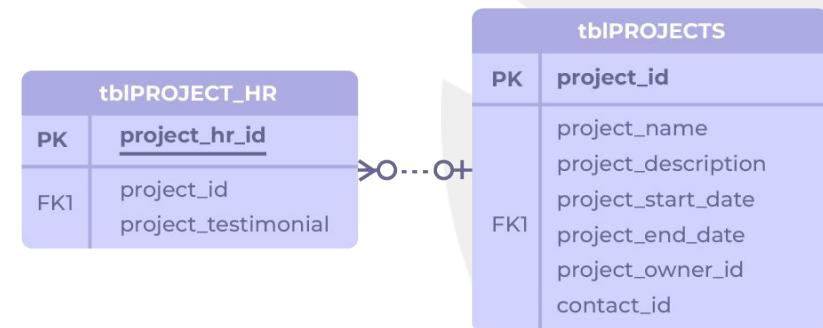


Zero or Many

### Berikut contoh dari relasinya~

Kamu masih bingung? Simak penjelasannya di bawah ini, yaaa~

- Table project\_hr punya foreign key project\_id
- Pada table project\_hr bisa punya row dengan project\_id yang sama.  
Misal terdapat 10 row dengan project\_id 234
- Tapi, bisa jadi ada project\_id yang nggak ada di project\_hr



Misalnya kita mau mengambil nama suatu project dan testimoni dari suatu project.

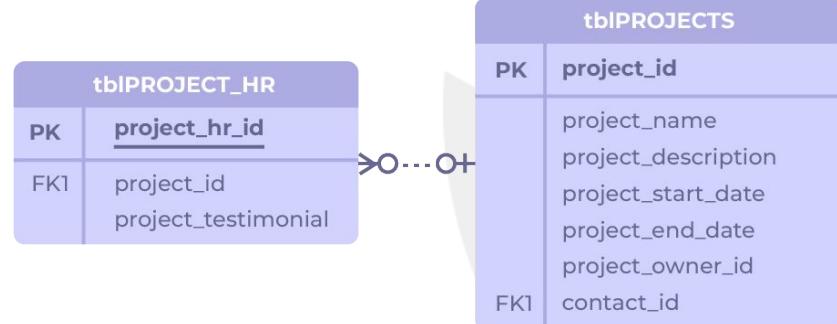
Cek query berikut ya!

```
Select p.project_name as nama_proyek, ph.project testimonial as testimoni  
from projects p  
join project_hr ph on ph.project_hr = p.project_id
```

## Dari query tersebut kita bisa ambil project dan testimoninya

Tapi, yang ditampilkan cuma project yang punya testimoni aja.

**Di query kayak gini kita sebaiknya menggunakan “alias” dari sebuah table**, karena kalau nggak menggunakan alias, kita harus menuliskan nama table-nya secara lengkap.



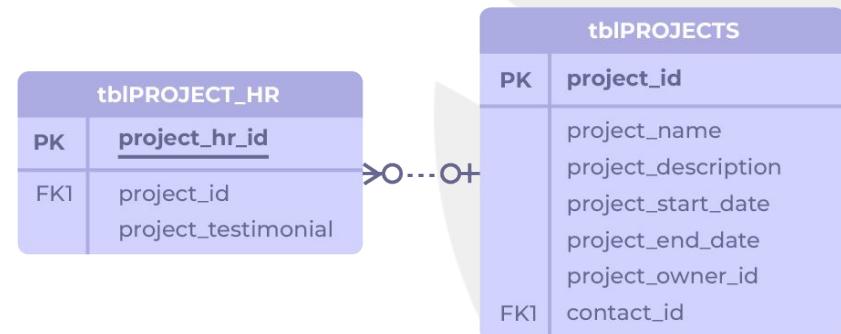
Lalu, apakah mungkin melakukan filtering dengan suatu spesifikasi?

Buat tahu jawabannya, coba kamu cek query berikut ya!



```
Select p.project_name as nama_proyek, ph.project testimonial as testimoni  
from projects p  
join project_hr ph on ph.project_hr = p.project_id  
where p.project_owner_id = 234
```

Dengan query yang udah kamu bikin sebelumnya, data yang ditampilkan cuma row dengan project\_owner\_id yang bernilai 234 aja, gengs.



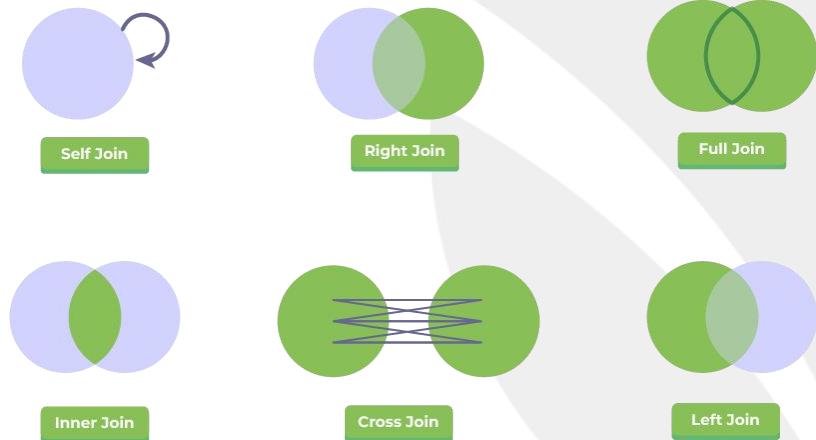
### Kita udah kenal join dan left join. Sekarang perhatikan gambar di samping, ya!

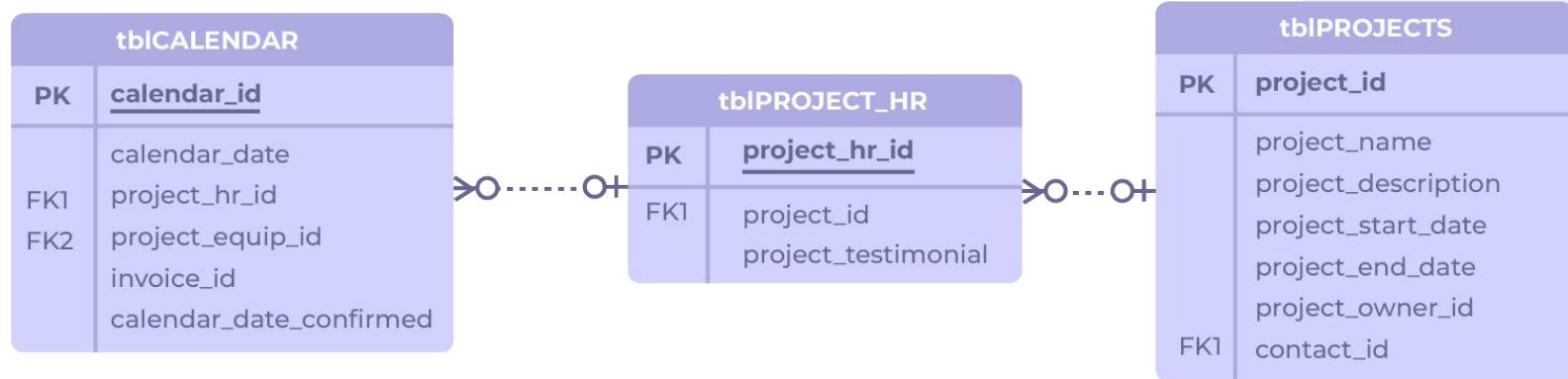
Gambar di samping merupakan gambaran dari seluruh join di PostgreSQL.

Join yang kita kenal sebelumnya sama dengan inner join, dimana kita bisa menggunakan join aja tanpa harus menggunakan prefix inner.

Untuk join yang lainnya bisa dipelajari di bawah ini yaaaaa~

#### Joins in PostgreSQL





Misalnya table calendar dan project\_hr merupakan table yang bakal terisi kalau project udah berjalan.

Disini kita mau menampilkan project\_name, project\_testimonial, dan invoice\_id

Kalau misalnya table calendar dan project\_hr merupakan table yang bakal terisi kalau project udah berjalan, lalu kita mau menampilkan project\_name, project\_testimonial dan invoice\_id maka kita bisa melakukan query sebagai berikut:

```
SELECT p.project_name, ph.project_testimonial, c.invoice_id  
from projects p  
join project_hr ph on ph.project_hr = p.project_id  
join calendar c on c.project_hr_id = ph.project_hr_id
```

# LANJUTTT



Sekarang kamu udah ada bayangan tentang Query?

Biar makin detail nih pemahamannya, kita bakal kupas tuntas tentang **Aggregate Query**.

### Aggregate Query itu apa yaaa?

Setelah tahu dasar-dasar query kayak join, kita bakal bahas lebih jauh tentang aggregate query.

Aggregate query merupakan query dengan aggregate function yang bakal **menghimpun dan mengolah data yang ada** sehingga nantinya ada data yang lebih sedikit setelah data tersebut diolah.



### Aggregate query pakai beberapa function nih, sob~

Berikut adalah functionnya:

- **MIN()**
- **MAX()**
- **DISTINCT()**
- **SUM()**

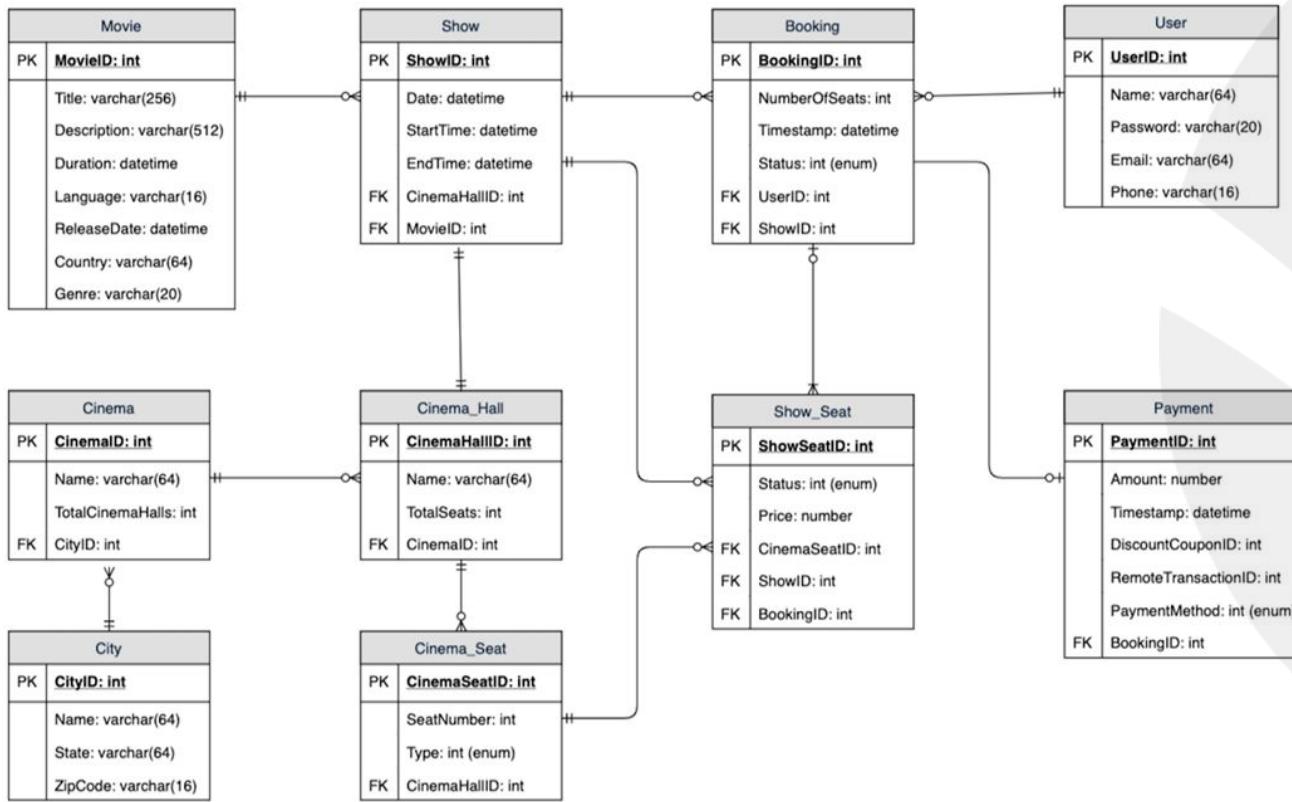


Selain pakai aggregate function tersebut, biasanya query ini juga pakai GROUP BY clause, yang bakal bantu dalam mengelompokkan datanya.

Buat menjelaskan aggregate query, kita bakal pakai ERD di slide selanjutnya, ya!



# Database Operation



### MIN() dan MAX()

Query dengan MIN() dan MAX() dipakai untuk menampilkan nilai minimum atau maksimum dari suatu kolom.

Penggunaannya gampang banget, kok. Berikut contohnya:

- Buat menampilkan harga termahal

```
select MIN(price) from show_seat;
```

- Buat menampilkan harga termurah

```
select MAX(price) from show_seat;
```

Show_Seat	
PK	ShowSeatID: int
	Status: int (enum)
	Price: number
FK1	CinemaSeatID: int
FK1	ShowID: int
FK1	BookingID: int

- Buat menampilkan harga termahal dari setiap ShowID

```
select MIN(price), showID from show_seat GROUP BY  
showID;
```

- Buat menampilkan harga termurah dari setiap ShowID

```
select MAX(price), showID from show_seat GROUP BY  
showID;
```

Show_Seat	
PK	ShowSeatID: int
	Status: int (enum)
	Price: number
FK1	CinemaSeatID: int
FK1	ShowID: int
FK1	BookingID: int

### DISTINCT()

Distinct merupakan aggregate function yang dipakai untuk menyatukan data yang terduplicasi sehingga nggak ada data yang berulang.

Di bawah ini ada contoh penggunaannya:

- Buat menampilkan ShowID tanpa data berulang

```
select DISTINCT>ShowID from show_seat
```

Show_Seat	
PK	ShowSeatID: int
	Status: int (enum)
	Price: number
FK1	CinemaSeatID: int
FK1	ShowID: int
FK1	BookingID: int

### SUM()

Query SUM() dipakai untuk melakukan penjumlahan. Berikut contoh penggunaannya:

- Untuk menghitung jumlah amount dari seluruh pembayaran dengan suatu method

```
select SUM(amount) from payment where  
paymentMethod = 234
```

- Untuk menghitung jumlah amount dari semua pembayaran masing-masing method

```
Select SUM(amount), paymentMethod from payment  
GROUP BY paymentMethod
```

Payment	
PK	PaymentID: int
FK1	Amount: number Timestamp: datetime DiscountCouponID: int RemoteTransactionID: int PaymentMethod: int (enum) BookingID: int

Sebelumnya kita udah bahas tentang dasar Query dan Aggregate Query. Selanjutnya kita bakal coba cari tahu tentang **Sub Query**.

“Sub Query tuh mirip kayak sub bab di dalam buku bukan, sih?”

Buat tahu jawabannya, yuk kita next slide!

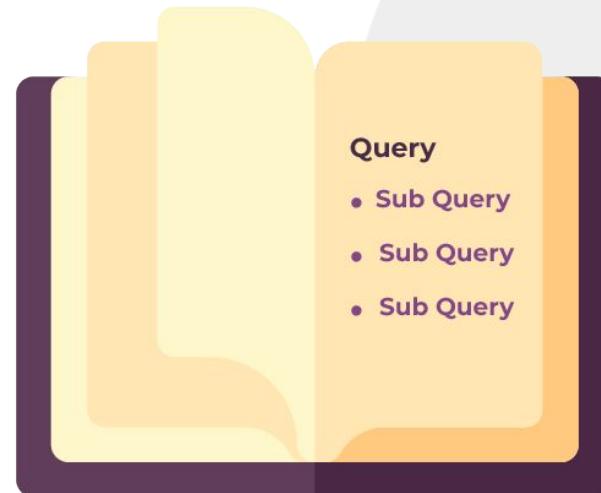


### Ibarat buku yang punya sub bab buat setiap materinya, Query juga sama, lho~

Yap, betul banget!

Sub query secara singkat merupakan **query di dalam query**. Kadang subquery dibutuhkan untuk mendapatkan data yang lebih spesifik.

Sub query juga bisa dibutuhkan untuk memperkecil scope data yang bakal diolah. Sehingga nantinya bisa meningkatkan performance query.



Tapiii...

Dalam penggunaan sub query, kita harus hati-hati, ya. Karena bisa saja performance dari sebuah query malah jadi lebih lambat nantinya.

Tentang performance query, ada beberapa hal yang bisa menyebabkan performance query jadi turun akibat kesalahan penulisan query, lho.

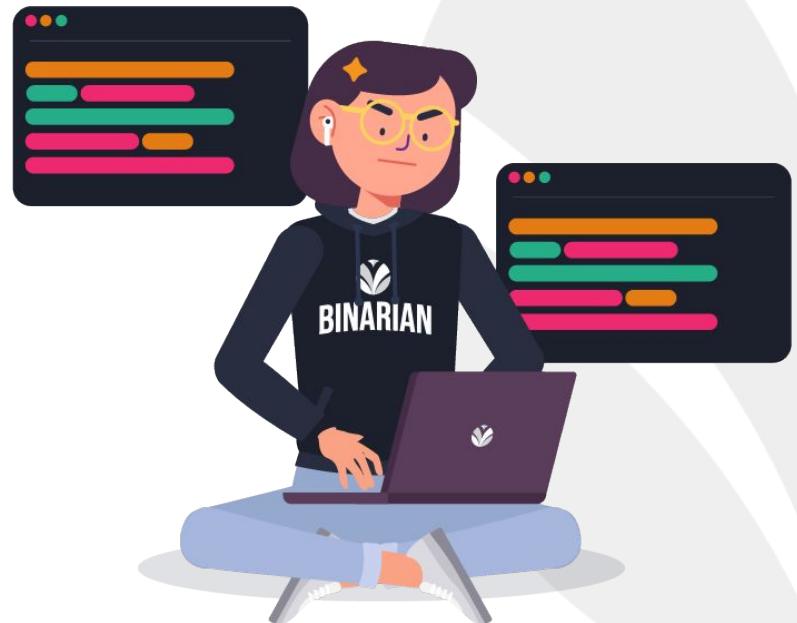
Kira-kira apa aja yaaa?



**Berikut adalah penyebab dari performance query yang jadi turun, sob!**

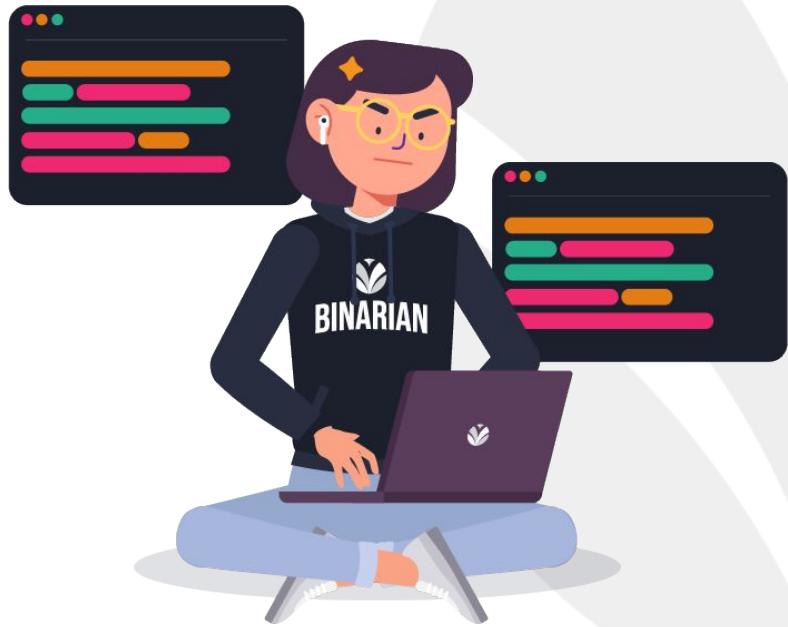
- 1. Terlalu banyak field yang nggak dipakai di subquery**

Pas pakai sub query, sebaiknya cuma pakai field yang dibutuhkan aja karena sub query bakal membentuk suatu virtual table yang bakal memakan memory.



### 2. Scope data jadi lebih besar pas melakukan sub query

Scope data diperbolehkan jadi besar kalau emang udah jadi kebutuhan. Tapi bisa jadi query yang salah malah menyebabkan pengambilan data jadi lebih besar.



### 3. Terlalu banyak memakai sorting seperti order by

Sorting pada query dipakai untuk mengurutkan data. Tapi sayangnya, sorting selalu bikin performance jadi lebih lambat.

Kalau nggak diperlukan, sebaiknya sorting nggak usah dilakukan.

Apalagi kalau dipakai di dalam subquery (diperbolehkan memakai sorting kalau ada tujuannya). Sebaiknya sorting dilakukan di query terluar (main query) aja, ya.



### Coba simak contoh penggunaan sub query, gengs~

Misalnya, kita mau mengambil data seluruh pembayaran yang dilakukan buat menonton film yang berbahasa Indonesia, nih. Contoh dari gambar berikut yang disebut sebagai sub query-nya.

```
SUM(p.amount)
from payment p
join booking b on b.bookingId = p.bookingId
join show s on b.showId = s.ShowId
where s.movieId in ( select movieId from movie where language = 'INDONESIA' )
Query select movieId from movie where language = 'INDONESIA'
```

Query tersebut juga ekuivalen dengan query di bawah.

Perhatikan baik-baik, ya!

```
● ● ●  
SUM(p.amount)  
from payment p  
join booking b on b.bookingId = p.bookingId  
join show s on b.showId = s.ShowId  
Join ( select movieId from movie where language = 'INDONESIA' ) movie m on  
m.movieId = s.movieId
```

Dasar Query, udah ✓

Aggregate Query, aman ☐□

Sub Query, beres □□

Next, kita bakal melakukan **Design Database** dengan ERD.

Yuk, langsung aja kita cari tahuu~



### Apa itu ERD?

**Entity Relationship Diagram** dikenal juga sebagai ERD, ER Diagram, atau model ER yang ditemukan pertama kali pada tahun 1976 oleh Peter Chen.

ERD ini adalah **jenis diagram struktural** yang dipakai untuk **mendesain database**.

Diagram ini dideskripsikan sebagai model yang menjelaskan **isi dan hubungan data** pada suatu database dalam bentuk **entitas, atribut**, serta **hubungan** antar entitas.



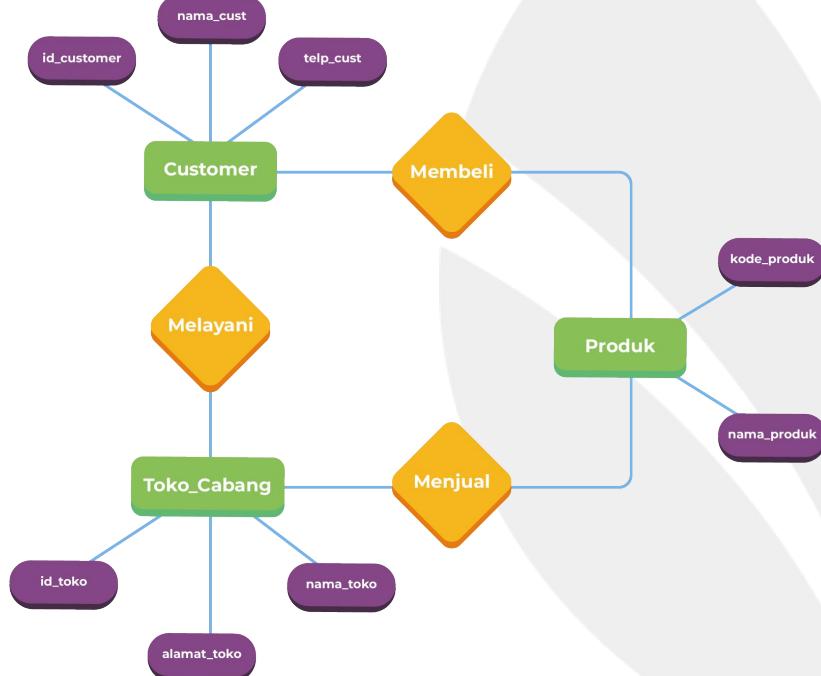
## ERD tuh kayak gini nih!

“ERD itu bentuknya kayak gimana?“

Coba kita perhatikan gambar disamping, ya!

ERD berisi **simbol** dan **konektor** yang menggambarkan dua informasi penting:

- **Entitas utama** dalam ruang lingkup sistem
- **Hubungan antar entitas** yang ada dalam sistem



### Wah, ERD memang sepenting itu, ya!

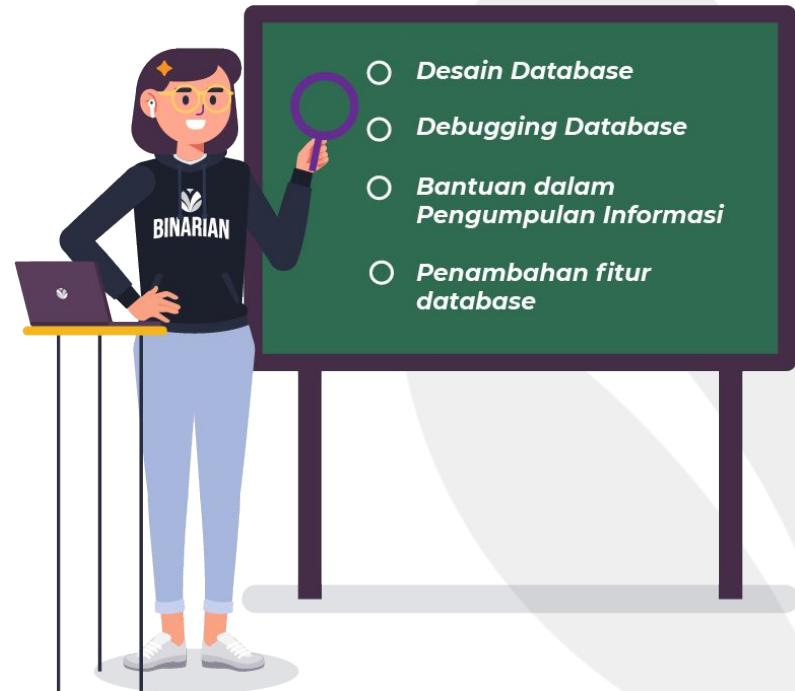
Selain bisa digunakan untuk **merancang arsitektur desain sebelum membangun database fisik**, sebenarnya ERD bisa dipakai untuk beberapa hal lain.

Bisa buat apa aja sih? Yuk kita cari tahu!



Berikut ini adalah beberapa kasus penggunaan dan penerapan ERD:

- **Desain database**
- **Debugging database**
- **Bantuan dalam pengumpulan informasi**
- **Penambahan fitur database**



### ERD memang memudahkan kita ☐

Berikut adalah beberapa manfaat dari ERD:



**Memudahkan proses analisis** database/sistem dengan cara yang **cepat dan murah**.



Memungkinkan **pengujian model** tanpa harus melakukan **proses pada database**, cukup dengan menggambar ERD.



**Menjelaskan hubungan antar data** dalam database berdasarkan objek-objek data yang terhubung dengan suatu relasi.



**Mendokumentasikan data** dengan cara mengidentifikasi setiap entitas dari data-data dan hubungannya pada suatu ERD itu sendiri.

Dalam merancang ERD, kita harus melewati beberapa tahapan, gengs!

- **Tahap 1** – Tentukan **entitas** yang bakal terlibat atau menentukan tabel.
- **Tahap 2** – Tentukan **atribut** dari masing-masing entitas.
- **Tahap 3** – Tentukan **relationship** antar entitas.
- **Tahap 4** – Buat **model ERD**.



### Tahap 1, tentukan entitas terlebih dahulu

Misalnya, kita mau bikin ERD dalam sistem database penjualan toko roti yang kita beri nama Sobi Roti.

Dalam sistem ini, aktivitas yang bakal terjadi adalah **Customer** membeli **Produk**, dan **Toko Cabang** menjual **Produk**.

Untuk sistem ini, berarti kita punya tiga entitas: **Customer**, **Toko\_Cabang**, dan **Produk**.



### Tahap 2, tentukan atribut masing-masing entitas

Dari ketiga entitas tadi, kita tentukan atributnya:

- **Customer:** id\_cust, nama\_cust, telp\_cust
- **Toko\_Cabang:** id\_toko, nama\_toko, alamat\_toko
- **Produk:** kode\_produk, nama\_produk



### Tahap 3, tentukan hubungan (relasi) antar entitas

Setelah menentukan atribut, maka langkah selanjutnya adalah menentukan relasi atau hubungan yang bakal terjadi antar entitas, misalnya:

- **Customer (One) membeli Produk (Many)**, yaitu satu customer beli banyak produk.
- **Toko\_Cabang (One) melayani Customer (Many)**, yaitu satu toko cabang melayani banyak customer.
- **Toko\_Cabang (One) menjual Produk (Many)**, yaitu satu toko cabang jual banyak produk.



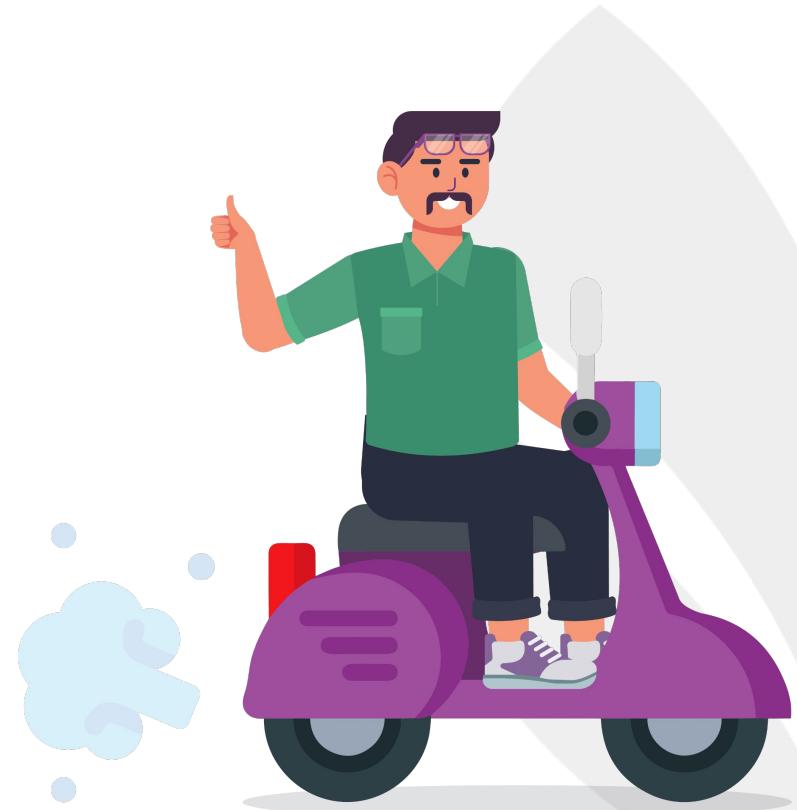
### Tahap 4, kita buat model ERD-nya!

Setelah menentukan relasi dari tiap entitas, kita bisa mulai merancang ERD-nya.

Dalam tahap ini, kita tinggal menggabungkan seluruh himpunan dan relasi dari tahap ketiga tadi dan membuat ilustrasinya.

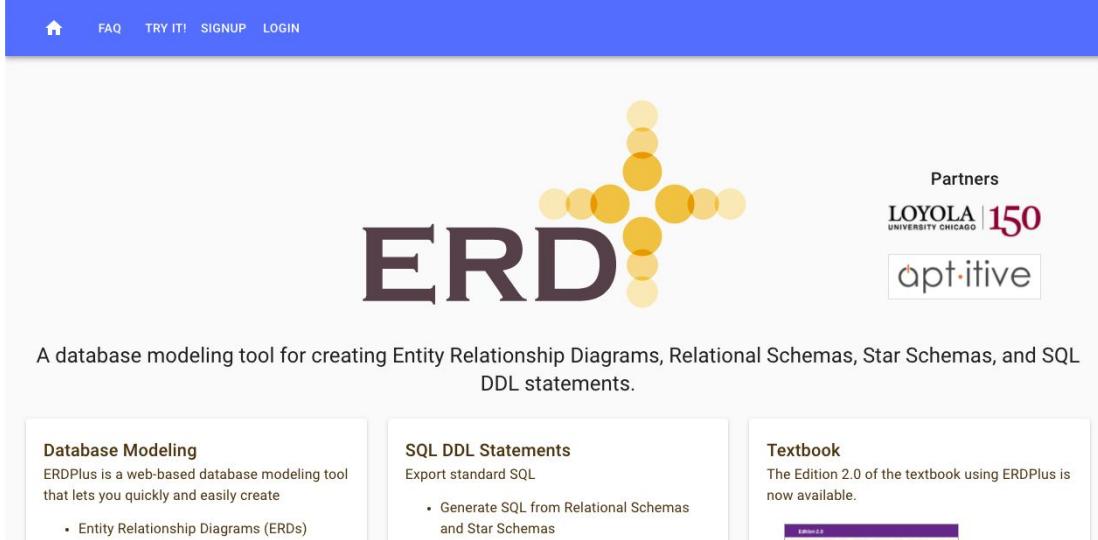
Di sinilah kita bakal pakai tools ERD, yaitu <https://erdplus.com/>

Yuk kita lihat step by step-nya!



Di halaman utama ERD Plus, klik pada menu “Try It!”

Menu tersebut bakal membawa kita ke halaman tempat kita bakal mempraktikkan desain ERD ini~



The screenshot shows the main interface of ERD Plus. At the top, there's a blue header bar with a house icon and the words "FAQ", "TRY IT!", "SIGNUP", and "LOGIN". Below the header is a large logo consisting of the word "ERD" in a dark grey sans-serif font, with a cluster of yellow circles of varying sizes above it, forming a stylized cross or asterisk shape. To the right of the logo, the word "Partners" is written in a smaller grey font, followed by two logos: "LOYOLA UNIVERSITY CHICAGO 150" and "opt.itive". Below the logo area, a descriptive text reads: "A database modeling tool for creating Entity Relationship Diagrams, Relational Schemas, Star Schemas, and SQL DDL statements." The main content area is divided into three sections: "Database Modeling" (with a description and a bullet point about Entity Relationship Diagrams), "SQL DDL Statements" (with a description and a bullet point about generating SQL from Relational and Star Schemas), and "Textbook" (with a description and a "Edition 2.0" button).

A database modeling tool for creating Entity Relationship Diagrams, Relational Schemas, Star Schemas, and SQL DDL statements.

**Database Modeling**  
ERDPlus is a web-based database modeling tool that lets you quickly and easily create

- Entity Relationship Diagrams (ERDs)

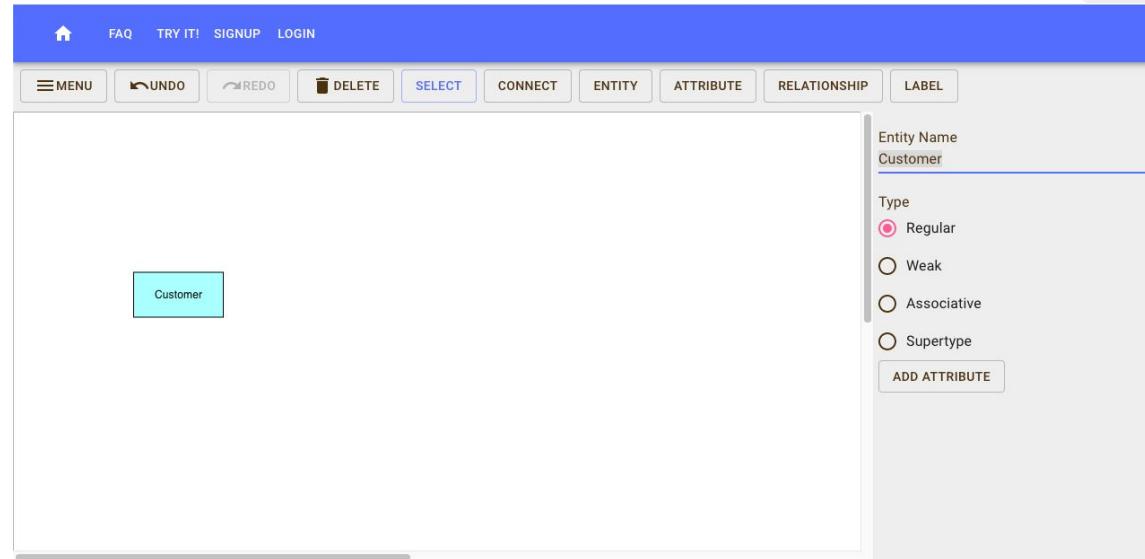
**SQL DDL Statements**  
Export standard SQL

- Generate SQL from Relational Schemas and Star Schemas

**Textbook**  
The Edition 2.0 of the textbook using ERDPlus is now available.

Edition 2.0

Kita bakal mulai dengan bikin entitas **Customer**. Klik pada bagian Entity, kasih nama sesuai nama entitas, yaitu Customer. Disini, kita berhasil membuat satu entitas!



Terus, **beri atribut** pada entitas Customer.

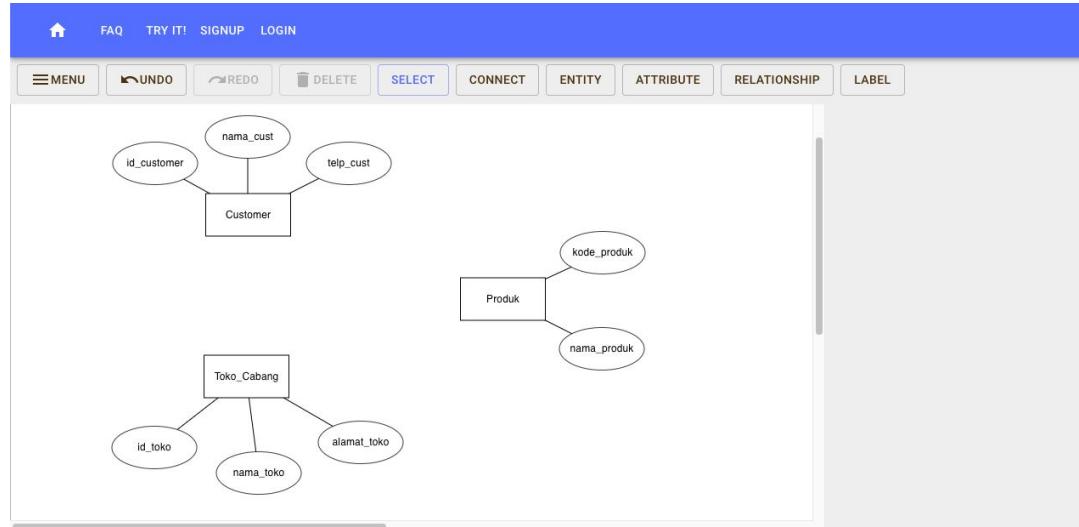
Klik di bagian **Add Attributes**, kasih nama sesuai atribut yang menempel pada entitas Customer, yaitu **id\_cust**, **nama\_cust**, dan **telp\_cust**.

Kita bisa menata tampilan entitas dan atribut cuma dengan drag and drop pada setiap objek supaya lebih rapi.



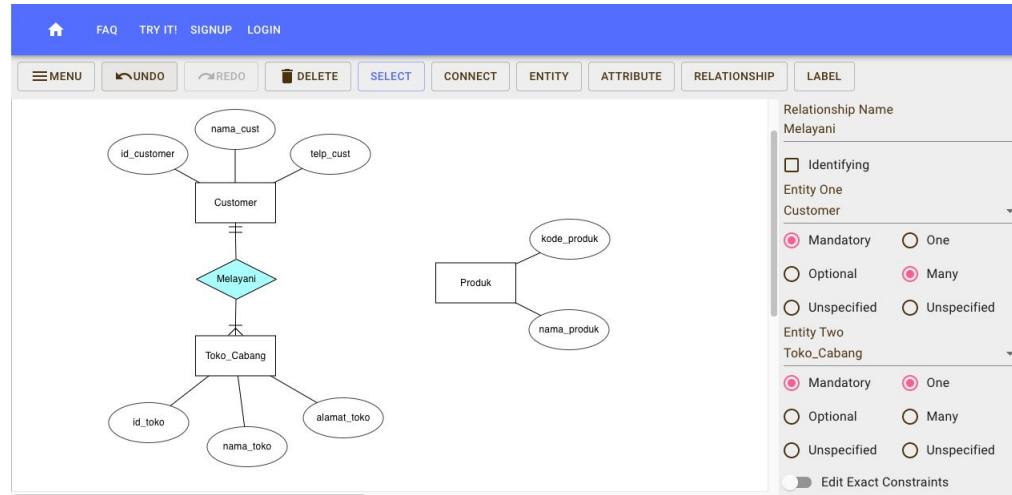
Ulangi langkah-langkah tadi untuk membuat semua entitas dan atribut sesuai kebutuhan kita. Kalau udah, tampilannya bakal terlihat seperti pada gambar dibawah ini.

Taraaaa~



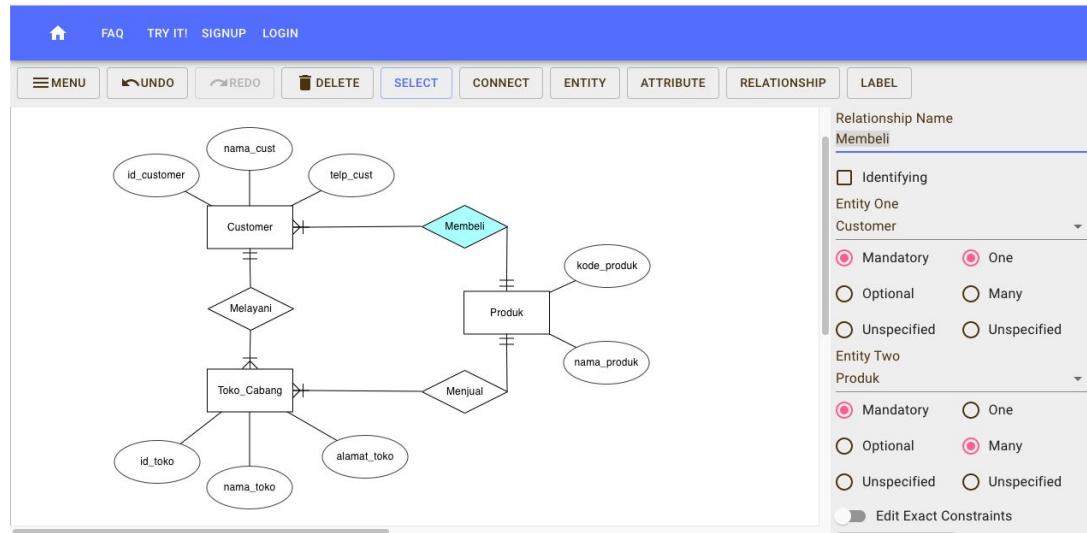
Langkah terakhir, hubungkan relasi antar entitas. Kita melakukan ini dengan pakai menu **Connect**. Klik menu Connect, terus klik dan tahan entitas Customer, lalu drag ke entitas lain yang kita mau, misalnya **Toko\_Cabang**. Selanjutnya, masukkan nama relasi (Relationship Name) sesuai dengan relasi yang kita buat, yaitu **Toko\_Cabang Melayani Customer**.

Jangan lupa buat klik **Many** pada bagian entitas Customer, dan **One** pada entitas Toko\_Cabang. Set keduanya di **Mandatory**.



Lakukan hal serupa pada relasi Customer Membeli Produk, dan relasi Toko\_Cabang Menjual Produk. Kalau udah, tampilannya bakal terlihat seperti pada gambar. Voila, kita udah berhasil bikin rancangan ERD!

Gampang kan gengs? ✌



Kita udah ada di materi terakhir di topic ini, yaitu tentang **Normalization**.

Sebagaimana artinya, normalization nggak akan jauh jauh dari data yang dianggap normal.

Tapi, kayak apa sih makna normal disini?



### So, apa itu Normalization?

Database Normalization adalah **pendekatan suatu desain database supaya database punya struktur data yang normal**.

“Terus, data yang normal itu gimana?”

Data yang normal adalah data yang nggak berulang. **Kondisi yang nggak normal bisa mengakibatkan sulitnya melakukan manipulasi data** kayak insert, update dan delete.



**Setidaknya, ada 9 bentuk normal data lho,  
gengs~**

Dari sembilan bentuk normal ini kita cuma membahas 3 contoh yang lazim dipakai aja, yaitu 1NF, 2 NF, 3NF.

Oh iya, NF itu artinya normal form. Contohnya sebagai berikut:

- [DBMS 1NF](#)
- [DBMS 2NF](#)
- [DBMS 3NF](#)

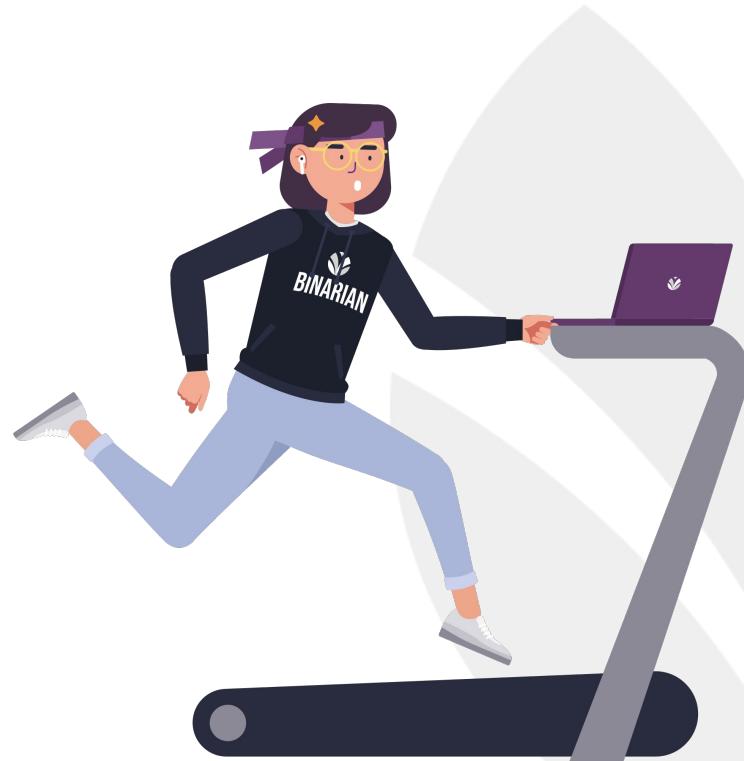


### Bip bip~

Ada misi buat kamu nih supaya makin ajib belajar database.

**Silakan buatlah design database sederhana dengan ERD.**

Nanti kalau udah selesai, silakan diskusikan jawabannya bersama teman sekelas dan facilitator, ya!

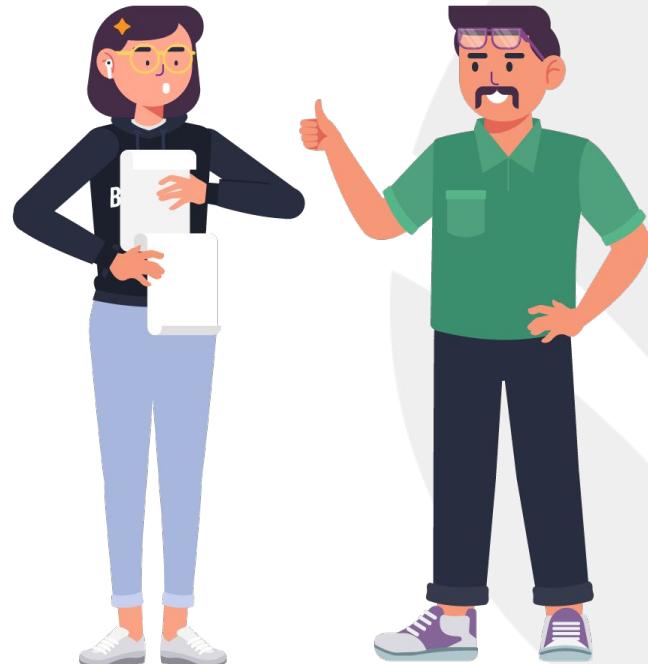


### Tadaa, ada penugasan lagi, nih!

Gimana latihannya? Apakah membantu kamu untuk lebih paham tentang database?

Nah, selanjutnya kamu ada tugas yang perlu diselesaikan. Di mana kamu perlu **Membuat ERD beserta CRUD operationnya**.

Kalau tugasnya selesai, kamu boleh minta facilitator untuk mengecek penugasan yang kamu kerjakan di pertemuan selanjutnya. Selamat mengerjakan ☐



Kalo kita recall, **subquery** dibutuhkan untuk mendapatkan data yang lebih spesifik walaupun resikonya adalah performance dari query tersebut malah jadi lebih lambat nantinya.

Kira-kira, pada case apa aja ya kita perlu untuk menggunakan subquery?



Nah, selesai sudah pembahasan kita di Chapter 4 Topic 3 ini.

Selanjutnya, kita bakal bahas tentang NoSQL.

Penasaran kayak gimana? Yuk langsung ke topik selanjutnya~

