JOINS

Wat is een Join?

Een **join** in MySQL stelt ons in staat om data uit meerdere tabellen te combineren. Dit is vooral handig als de tabellen op een bepaalde manier aan elkaar gerelateerd zijn via een primaire en vreemde sleutel (foreign key). Voor de wagens-database in jouw voorbeeld hebben we twee tabellen: `wagens merken` en `wagens modellen`.

- **wagens merken**: Bevat informatie over verschillende automerken zoals Audi, BMW, enz.
- **wagens modellen**: Bevat informatie over verschillende modellen die bij elk merk horen, met een `merkid` die aangeeft bij welk merk het model hoort.

De sleutelrelatie tussen de tabellen stelt ons in staat gegevens uit beide tabellen te combineren, zoals welk model bij welk merk hoort. Laten we de verschillende soorten joins bekijken die in MySQL beschikbaar zijn.

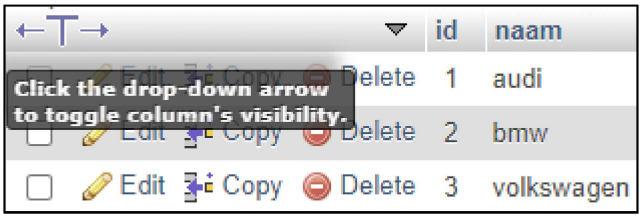
Types van Joins

MySQL ondersteunt vier hoofdtypes van joins: 1. **INNER JOIN** 2. **LEFT JOIN** 3. **RIGHT JOIN** 4. **CROSS JOIN**

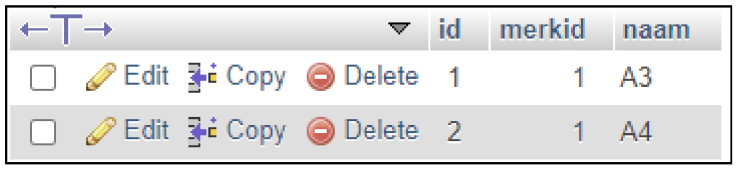
We zullen elk type join stap voor stap behandelen, met uitleg en voorbeelden van de query's en de resulterende output.

We vullen de 2 tabellen als volgt:

merken:



modellen:



1. INNER JOIN

Een **INNER JOIN** geeft alleen de rijen weer die in beide tabellen een overeenkomst hebben. Dit betekent dat alleen de merken die ook modellen hebben, worden weergegeven.

Voorbeeld:

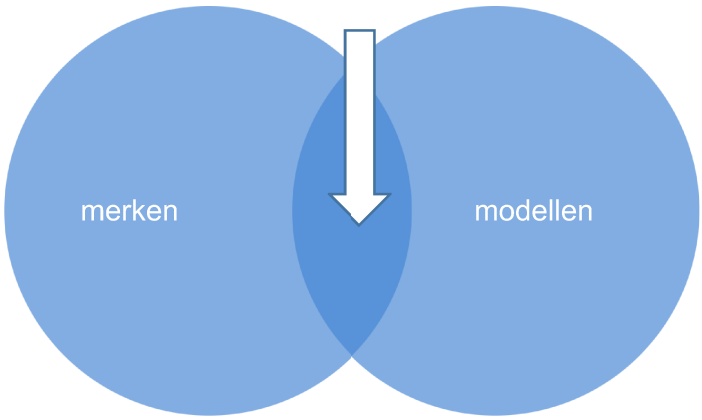
```
SELECT merken.naam, modellen.naam
FROM merken
INNER JOIN modellen ON merken.id = modellen.merkid;
```

- Hier verbinden we de tabel `merken` met de tabel `modellen` op basis van de `id` van `merken` en de `merkid` van `modellen`.
- Het resultaat laat alleen merken en hun bijbehorende modellen zien, zoals Audi met A3 en A4.

naam	naam
audi	A3
audi	A4

Grafisch:

Een Venn-diagram voor een **INNER JOIN** toont alleen de overlappende gedeelten tussen `merken` en `modellen`.



KLASSIKALE OEFENINGEN

SAKILA DATABASE

1. **Toon alle films met de acteurs en de datum waarop elke film beschikbaar werd gesteld**

   ```
   SELECT film.title, actor.first_name, actor.last_name, film.release_year
   FROM film
   INNER JOIN film_actor ON film.film_id = film_actor.film_id
   INNER JOIN actor ON film_actor.actor_id = actor.actor_id;
   ```

   – Hier wordt de release-informatie van de film gecombineerd met de acteurs die in de film spelen.

2. **Vind alle medewerkers met de huurtransacties die zij persoonlijk hebben afgehandeld**

   ```
   SELECT staff.first_name, staff.last_name, rental.rental_date,
   rental.return_date
   FROM staff
   INNER JOIN rental ON staff.staff_id = rental.staff_id;
   ```

   – Deze query toont alle medewerkers met hun respectieve huurtransacties, wat helpt om te begrijpen wie welke klanten heeft geholpen.

2. LEFT JOIN

Een **LEFT JOIN** toont alle rijen uit de linkertabel (`merken`), ongeacht of er een overeenkomst is in de rechtertabel (`modellen`). Als er geen bijpassend model is, wordt er `NULL` weergegeven.

Voorbeeld:

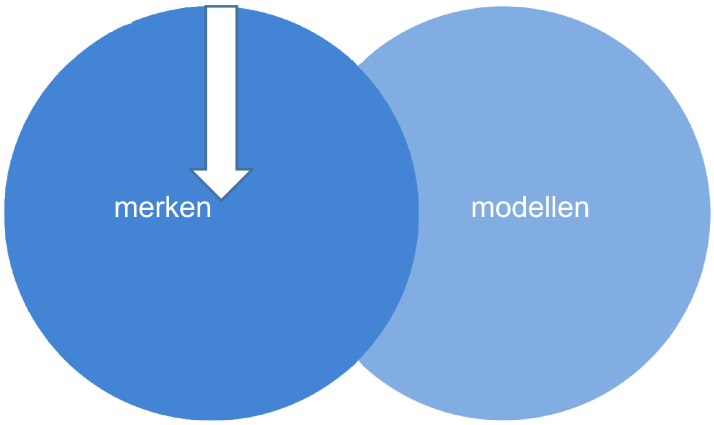
```
SELECT merken.naam, modellen.naam
FROM merken
LEFT JOIN modellen ON merken.id = modellen.merkid;
```

- Deze query selecteert alle merken, zelfs als ze geen modellen hebben. Bijvoorbeeld, als Volkswagen geen model heeft, wordt `NULL` weergegeven bij het model.

Resultaat:

naam	naam
audi	A3
audi	A4
bmw	NULL
volkswagen	NULL

Grafisch:



Het Venn-diagram voor een **LEFT JOIN** laat de volledige cirkel van `merken` zien, plus de gedeelten die overlappen met `modellen`.

Voorbeeld:Stel je hebt een lijst van klanten en een lijst van bestellingen. Je wilt een overzicht van alle klanten, ook degenen die nog geen bestelling hebben geplaatst.

KLASSIKALE OEFENINGEN

SAKILA DATABASE

3. **Geef een lijst van alle klanten en hun laatst bekeken film, inclusief klanten die nog geen film hebben bekeken**

```
SELECT customer.first_name, customer.last_name, film.title
FROM customer
LEFT JOIN rental ON customer.customer_id = rental.customer_id
LEFT JOIN inventory ON rental.inventory_id = inventory.inventory_id
LEFT JOIN film ON inventory.film_id = film.film_id
ORDER BY rental.rental_date DESC;
```

   – Deze query toont alle klanten en, indien beschikbaar, de laatst gehuurde film per klant.

4. **Toon alle categorieën van films en de hoeveelheid films die eraan zijn gekoppeld**

```
SELECT category.name, COUNT(film_category.film_id) AS aantal_films
FROM category
LEFT JOIN film_category ON category.category_id =
film_category.category_id
GROUP BY category.category_id;
```

   – Deze query geeft een lijst van alle categorieën met het aantal films dat eraan gekoppeld is, inclusief categorieën zonder films.

3. RIGHT JOIN

Een **RIGHT JOIN** is vergelijkbaar met een LEFT JOIN, maar nu worden alle rijen uit de rechtertabel (`modellen`) weergegeven, ongeacht of er een overeenkomst is in de linkertabel (`merken`). Dit is handig als je zeker wilt weten dat je alle modellen toont, ook al is er geen bijbehorend merk.

Voorbeeld:

```
SELECT merken.naam, modellen.naam
FROM merken
RIGHT JOIN modellen ON merken.id = modellen.merkid;
```

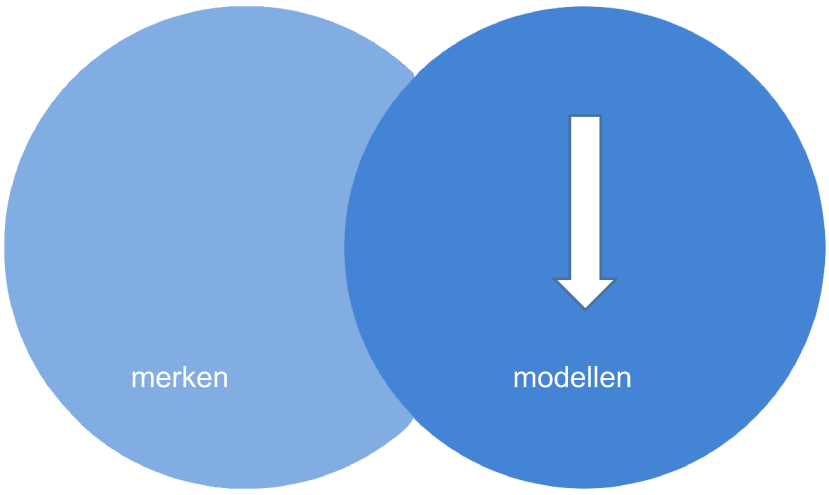
- In dit voorbeeld zouden alle modellen worden weergegeven, zelfs als er geen overeenkomstig merk is.

Resultaat:

Resultaat:

naam	naam
audi	A3
audi	A4
NULL	A3
NULL	A4

Grafisch:



Het Venn-diagram voor een **RIGHT JOIN** laat de volledige cirkel van `modellen` zien, plus de gedeelten die overlappen met `merken`.

Voorbeeld:Stel je hebt een lijst van bestellingen en een lijst van producten. Je wilt een overzicht van alle producten, ook als ze nooit zijn besteld.

KLASSIKALE OEFENINGEN

SAKILA DATABASE

5. **Toon een lijst van alle inventarisitems met de bijbehorende film, zelfs als er geen inventaris is gekoppeld aan een film**

```
SELECT inventory.inventory_id, film.title
FROM film
RIGHT JOIN inventory ON film.film_id = inventory.film_id;
```

   – Dit toont alle inventarisitems en de gekoppelde film, zelfs als een film momenteel niet in de inventaris zit.

6. **Vind alle steden met winkels en de namen van die winkels, inclusief steden zonder winkels**

```
SELECT city.city, store.store_id
FROM city
RIGHT JOIN address ON city.city_id = address.city_id
RIGHT JOIN store ON address.address_id = store.address_id;
```

   – Hiermee wordt een lijst gegenereerd van alle steden en, indien aanwezig, de winkels in die stad. Dit kan nuttig zijn voor inzicht in regio's zonder winkels.

4. CROSS JOIN

Een **CROSS JOIN** verbindt elke rij van de ene tabel met elke rij van de andere tabel. Het resultaat is een Cartesian product, wat betekent dat het aantal rijen het product is van het aantal rijen in beide tabellen. Dit type join is vaak inefficiënt voor grote datasets, maar kan nuttig zijn voor specifieke analyses.

Voorbeeld:

```
SELECT *
FROM merken
CROSS JOIN modellen;
```

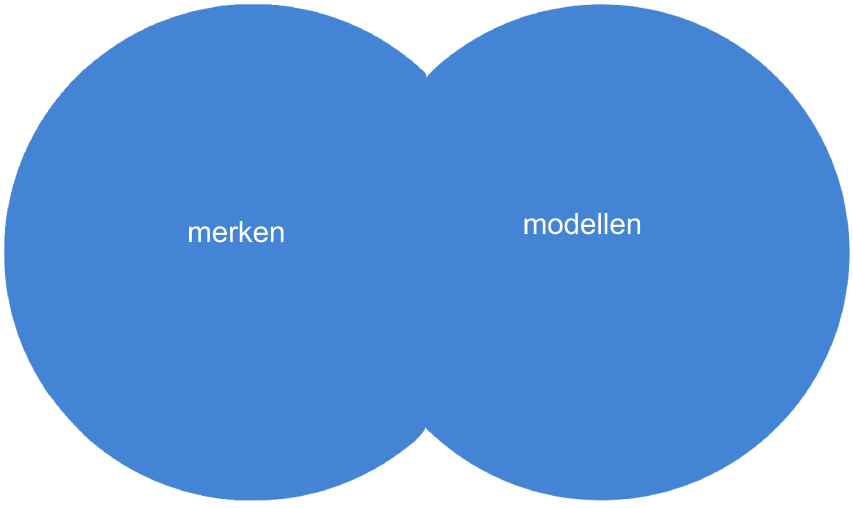
- Deze query combineert elke rij van de tabel `merken` met elke rij van de tabel `modellen`.

Resultaat:

Resultaat:

id	naam	id	merkid	naam
1	audi	1	1	A3
2	bmw	1	1	A3
3	volkswagen	1	1	A3
1	audi	2	1	A4
2	bmw	2	1	A4
3	volkswagen	2	1	A4

Grafisch:



Het Venn-diagram voor een **CROSS JOIN** toont de volledige overlapping tussen de twee tabellen, wat betekent dat elke rij in `merken` wordt gecombineerd met elke rij in `modellen`.

Voorbeeld:Stel je hebt een winkel met een lijst van klanten en een lijst van promotieproducten. Je wilt alle mogelijke combinaties van klanten en producten bekijken om te bepalen welke klanten je een promotie-aanbieding wilt sturen. In dit geval kun je een CROSS JOIN gebruiken om elk product met elke klant te combineren.

klanten tabel:

klant_id	naam
1	Jan
2	Maria
3	Peter

producten tabel:

product_id	naam
1	Koffiemok
2	T-shirt

Resultaat:

klant	product
Jan	Koffiemok
Jan	T-shirt
Maria	Koffiemok
Maria	T-shirt
Peter	Koffiemok
Peter	T-shirt

KLASSIKALE OEFENINGEN

SAKILA DATABASE

7. **Geef een combinatie van alle klanten en alle medewerkers**

```
SELECT customer.first_name AS klant, staff.first_name AS medewerker
FROM customer
CROSS JOIN staff;
```

   - Deze query toont alle mogelijke klant-medewerkercombinaties, wat nuttig kan zijn voor het evalueren van de mogelijke klant-ondersteuning relaties.

8. **Lijst van alle films en alle acteurs om mogelijke cast-opties te onderzoeken**

```
SELECT film.title, actor.first_name, actor.last_name
FROM film
CROSS JOIN actor;
```

   - Deze query toont een volledige lijst van elke mogelijke combinatie van films en acteurs, wat handig kan zijn bij het simuleren van potentiële castingkeuzes.

Samenvatting

- **INNER JOIN**: Toont alleen de rijen die overeenkomsten hebben in beide tabellen.
- **LEFT JOIN**: Toont alle rijen uit de linkertabel, zelfs als er geen overeenkomst is in de rechtertabel.
- **RIGHT JOIN**: Toont alle rijen uit de rechtertabel, zelfs als er geen overeenkomst is in de linkertabel.
- **CROSS JOIN**: Verbindt elke rij van de ene tabel met elke rij van de andere tabel.

Door deze join types goed te begrijpen, kun je effectief data combineren en relaties tussen tabellen inzichtelijk maken. Dit is een essentieel onderdeel van databasebeheer en helpt bij het opstellen van rijke datasets voor analyses en rapportages.

5. COMPLEXE OEFENINGEN

KLASSIKALE OEFENINGEN

SAKILA DATABASE

9. **Geef een lijst van films die zijn gehuurd door klanten, met de naam van de medewerker die de verhuur heeft uitgevoerd en de naam van de stad waarin de klant woont**

```
SELECT film.title, customer.first_name AS klant, staff.first_name AS
medewerker, city.city AS woonplaats
FROM film
INNER JOIN inventory ON film.film_id = inventory.film_id
INNER JOIN rental ON inventory.inventory_id = rental.inventory_id
INNER JOIN customer ON rental.customer_id = customer.customer_id
INNER JOIN address ON customer.address_id = address.address_id
INNER JOIN city ON address.city_id = city.city_id
INNER JOIN staff ON rental.staff_id = staff.staff_id;
```

   - Deze complexe join combineert meerdere tabellen om te tonen welke films door klanten zijn gehuurd, met de medewerkers die de transactie hebben afgehandeld en de stad waarin de klant woont.

10. **Toon het totaalbedrag dat klanten hebben betaald, gegroepeerd per stad**

```
SELECT city.city, SUM(payment.amount) AS totale_omzet
FROM payment
INNER JOIN rental ON payment.rental_id = rental.rental_id
INNER JOIN customer ON rental.customer_id = customer.customer_id
INNER JOIN address ON customer.address_id = address.address_id
INNER JOIN city ON address.city_id = city.city_id
GROUP BY city.city;
```

- Hier wordt het totale bedrag dat klanten hebben betaald per stad berekend, wat handig kan zijn voor marketinganalyse.