Backendprogrammering 1

# RELATIONSDATABASER

Utbildare: Mikael Olsson

mikael.olsson@emmio.se

076-174 90 43

NACKADEMIN

# Alias

- ```
  SELECT Person.FirstName,
  Person.LastName
  FROM Person
  ```

- ```
  SELECT p.FirstName, p.LastName
  FROM Person AS p
  ```

# JOIN

- SQL Server join :- Inner join, Left join, Right join and full outer join
  https://www.youtube.com/watch?v=KTvYHEntvn8
- MySQL har inte stöd för `FULL OUTER JOIN`, vi återkommer till en workaround senare.

# JOIN

Person

| PersonID | Fnamn | Enamn | Bostad | Bil |
|---|---|---|---|---|
| 1 | Eva | Vik | 1 | 1 |
| 2 | Sten | Vik | 1 | 1 |
| 3 | Fredrik | Vik | 1 | |
| 4 | Stina | Nilsson | 2 | 2 |
| 5 | Niklas | Nilsson | 2 | 3 |

Bostad

| BostadID | Adress | Rum |
|---|---|---|
| 1 | Vägen 1 | 3 |
| 2 | Gatan 3 | 1 |

# JOIN

```
SELECT *
  FROM Person P
    INNER JOIN Bostad B ON P.Bostad = B.BostadID
```

| Person | | | | | Bostad | | |
|---|---|---|---|---|---|---|---|
| **PersonID** | **Fnamn** | **Enamn** | **Bostad** | **Bil** | **BostadID** | **Adress** | **Rum** |
| 1 | Eva | Vik | 1 | 1 | 1 | Vägen 1 | 3 |
| 2 | Sten | Vik | 1 | 1 | 1 | Vägen 1 | 3 |
| 3 | Fredrik | Vik | 1 | | 1 | Vägen 1 | 3 |
| 4 | Stina | Nilsson | 2 | 2 | 2 | Gatan 3 | 1 |
| 5 | Niklas | Nilsson | 2 | 3 | 2 | Gatan 3 | 1 |

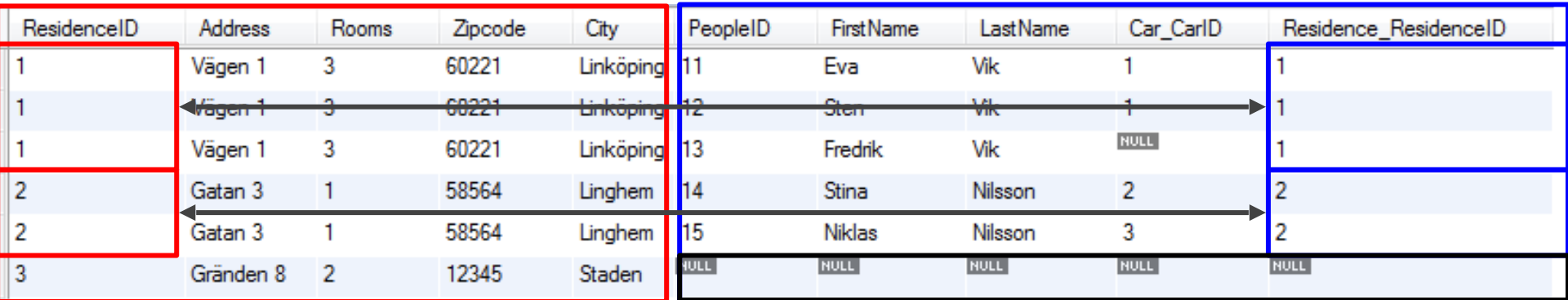
# JOIN

- Vad händer om vi lägger till en rad i Residence?

| ResidenceID | Address | Rooms | Zipcode | City |
|---|---|---|---|---|
| 1 | Vägen 1 | 3 | 60221 | Linköping |
| 2 | Gatan 3 | 1 | 58564 | Linghem |
| 3 | Gränden 8 | 2 | 12345 | Staden |

# OUTER JOIN

```
SELECT *
FROM Residence R
   LEFT OUTER JOIN People P ON
R.ResidenceID = P.Residence_ResidenceID
```



| ResidenceID | Address | Rooms | Zipcode | City | PeopleID | FirstName | LastName | Car_CarID | Residence_ResidenceID |
|---|---|---|---|---|---|---|---|---|---|
| 1 | Vägen 1 | 3 | 60221 | Linköping | 11 | Eva | Vik | 1 | 1 |
| 1 | Vägen 1 | 3 | 60221 | Linköping | 12 | Sten | Vik | 1 | 1 |
| 1 | Vägen 1 | 3 | 60221 | Linköping | 13 | Fredrik | Vik | NULL | 1 |
| 2 | Gatan 3 | 1 | 58564 | Linghem | 14 | Stina | Nilsson | 2 | 2 |
| 2 | Gatan 3 | 1 | 58564 | Linghem | 15 | Niklas | Nilsson | 3 | 2 |
| 3 | Gränden 8 | 2 | 12345 | Staden | NULL | NULL | NULL | NULL | NULL |

# DISTINCT

- Tar bort alla dubletter.

# UNION, INTERSECT, MINUS

- SQL `Intersect`, `Union`, `Union All`, `Minus`, and `Except`
  https://www.youtube.com/watch?v=bL5UX-p1wMc

# UNION - FULL OUTER JOIN

```
SELECT column_list
UNION [DISTINCT | ALL]
SELECT column_list
```

# INTERSECT

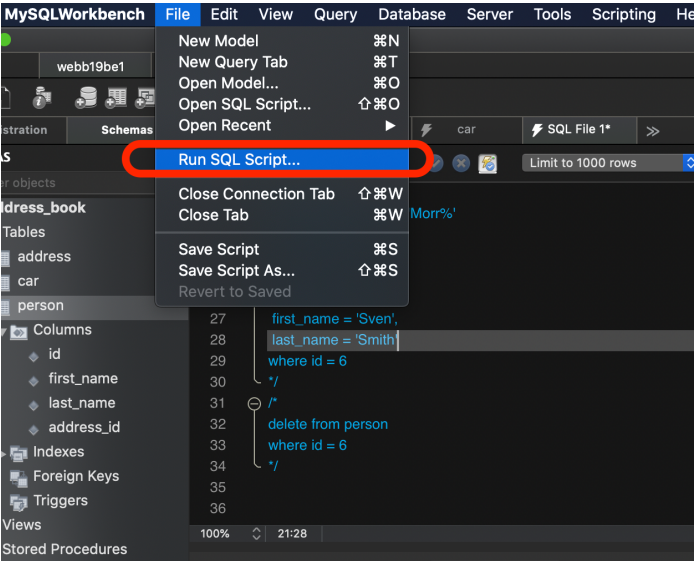
- Simulate MySQL `INTERSECT` operator using `DISTINCT` operator and `INNER JOIN` clause.
- 
```
SELECT DISTINCT id
FROM t1
    INNER JOIN t2 USING(id);
```

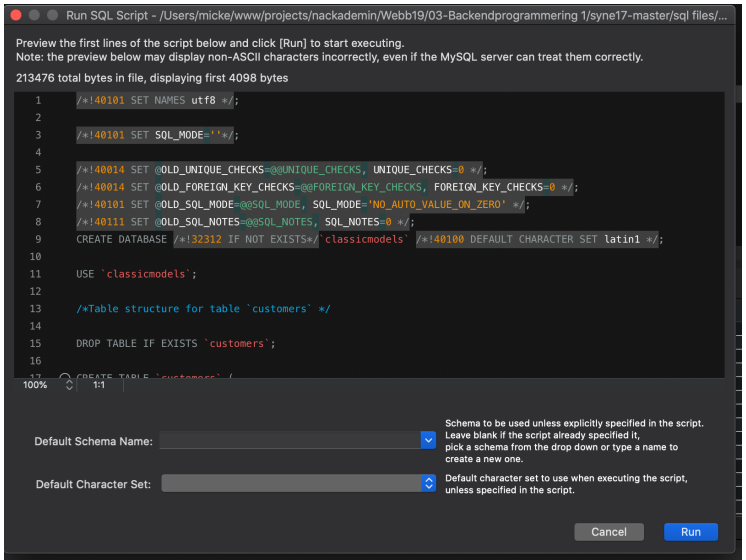
# MINUS

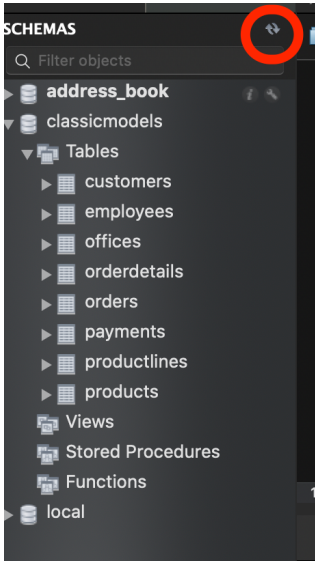
```
SELECT id
FROM t1
  LEFT JOIN t2 USING (id)
WHERE
  t2.id IS NULL;
```

# Exempeldatabas - Classic Models

- Finns på studentportalen



# Övning - Join

# Views

- Virtuell tabell
- Kan innehålla villkor och beräkningar
- https://www.youtube.com/watch?v=OP6zvaRdkuw

# Views exempel

- Publicerade artiklar
- Beräknat fält

# Egen-relationer

Vem är vems chef?

```
1  SELECT * FROM classicmodels.employees;
```

100% 1:1

Result Grid | Filter Rows: Search | Edit: | Export/Import:

| employeeNumber | lastName | firstName | extension | email | officeCode | reportsTo | jobTitle |
|---|---|---|---|---|---|---|---|
| 1002 | Murphy | Diane | x5800 | dmurphy@classicmodelcars.com | 1 | NULL | President |
| 1056 | Patterson | Mary | x4611 | mpatterso@classicmodelcars.com | 1 | 1002 | VP Sales |
| 1076 | Firrelli | Jeff | x9273 | jfirrelli@classicmodelcars.com | 1 | 1002 | VP Marketing |
| 1088 | Patterson | William | x4871 | wpatterson@classicmodelcars.com | 6 | 1056 | Sales Manager (APAC) |
| 1102 | Bondur | Gerard | x5408 | gbondur@classicmodelcars.com | 4 | 1056 | Sale Manager (EMEA) |
| 1143 | Bow | Anthony | x5428 | abow@classicmodelcars.com | 1 | 1056 | Sales Manager (NA) |

# Egen-relationer

Det är fullt tillåtet att använda samma tabell flera gånger i samma SELECT-sats. Vad skulle man kunna ha för nytta av det?

Kommer ni ihåg hur man ger en tabell ett alias?

# Egen-relationer

Vilka attribut vill vi koppla ihop?

e1

```
1  SELECT employeeNumber, lastName, FirstName, reportsTo
2  FROM employees;
```

100% 16:1

Result Grid | Filter Rows: Search | Edit:

| employeeNumber | lastName | FirstName | reportsTo |
|---|---|---|---|
| 1002 | Murphy | Diane | NULL |
| 1056 | Patterson | Mary | 1002 |
| 1076 | Firrelli | Jeff | 1002 |
| 1088 | Patterson | William | 1056 |

e2

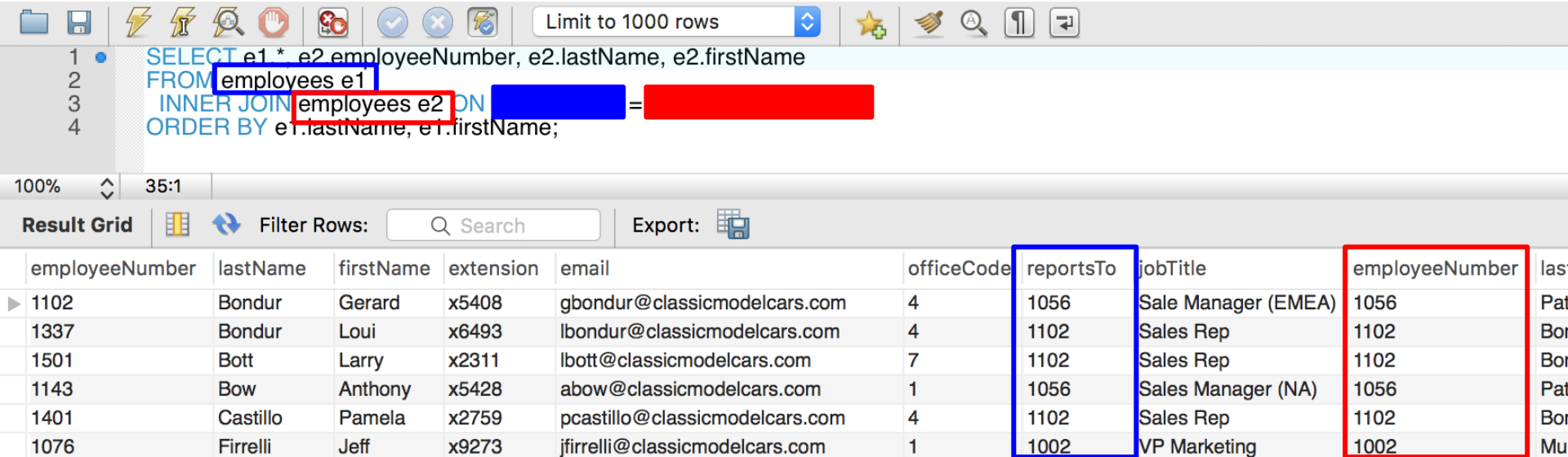
```
1  SELECT employeeNumber, lastName, FirstName, reportsTo
2  FROM employees;
```

100% 16:1

Result Grid | Filter Rows: Search | Edit:

| employeeNumber | lastName | FirstName | reportsTo |
|---|---|---|---|
| 1002 | Murphy | Diane | NULL |
| 1056 | Patterson | Mary | 1002 |
| 1076 | Firrelli | Jeff | 1002 |
| 1088 | Patterson | William | 1056 |

# Egen-relationer



# IN

```
SELECT *
FROM Car
WHERE owner = 1
  OR owner = 3
  OR owner = 8
```

```
SELECT *
FROM Car
WHERE owner IN (1, 3, 8)
```

http://www.mysqltutorial.org/sql-in.aspx

# Intro till funktioner

Det finns många inbyggda funktioner. Vissa kan vara lite olika i olika databaser.

```
SELECT CONCAT(firstName, ' ', lastName) AS fullName
FROM Person
```

```
SELECT IFNULL(lastName, '***') AS lastName
FROM Person
```

https://dev.mysql.com/doc/refman/5.7/en/func-op-summary-ref.html

# Mer funktioner

```
SELECT COUNT(*)
FROM products
```

Vad gör denna funktion?

# Mer funktioner

```
SELECT COUNT(*)
FROM products
WHERE productLine = ‘Motorcycle’
```

Kan man få ut antalet för varje productLine utan att göra en fråga för varje?

# GROUP BY

```
SELECT productLine, COUNT(*) AS NoOfItems
FROM products
GROUP BY productLine
```

https://www.youtube.com/watch?v=_uyyc5fc3J8

# Aggregerade funktioner

- AVG
- COUNT
- SUM
- MIN
- MAX

# HAVING

- Hur visar vi enbart rader där avgQuantityInStock > 3500?

```
1  SELECT productLine, AVG(quantityInStock) AS avgQuantityInStock
2  FROM products
3  GROUP BY productLine;
```

Result (1) Result (1) Result (1) Result (1) Result (1) Result (1) Result (1) Res

Fetched 7 records. Duration: 0.000

| productLine | avgQuantityInStock |
|---|---|
| Classic Cars | 5767.9737 |
| Motorcycles | 5338.5385 |
| Planes | 5190.5833 |
| Ships | 2981.4444 |
| Trains | 5565.3333 |
| Trucks and Buses | 3259.1818 |
| Vintage Cars | 5203.3333 |

# HAVING

## - WHERE funkar inte i aggregerade funktioner

```sql
SELECT productLine, AVG(quantityInStock) AS avgQuantityInStock
FROM products
WHERE avgQuantityInStock > 3500
GROUP BY productLine;
```

| | | |
|---|---|---|
| 249 | 22:12:10 | Error Code: 1054Unknown column 'avgQuantityInStock' in 'where clause' |

```sql
SELECT productLine, AVG(quantityInStock) AS avgQuantityInStock
FROM products
WHERE AVG(quantityInStock) > 3500
GROUP BY productLine;
```

| | | |
|---|---|---|
| 250 | 22:14:26 | Error Code: 1111Invalid use of group function |

# HAVING

```sql
SELECT productLine, AVG(quantityInStock) AS avgQuantityInStock
FROM products
GROUP BY productLine
HAVING avgQuantityInStock > 3500;
```

esult (1) Result (1) Result (1) Result (1) Result (1) Result (1) Result (1) Re

Fetched 5 records. Duration: 0.00

| productLine | avgQuantityInStock |
|---|---|
| Classic Cars | 5767.9737 |
| Motorcycles | 5338.5385 |
| Planes | 5190.5833 |
| Trains | 5565.3333 |
| Vintage Cars | 5203.3333 |

# HAVING

- `WHERE` används för att begränsa rader.
  - Används även för att avgöra vilka tabeller och index som ska användas.
- `HAVING` är ett “filter” på resultatet
  - Läggs på efter `ORDER BY` och `GROUP BY`.
- `WHERE` ger bättre performance än `HAVING`.

# Övning - Views & Aggregerade funktioner

Finns på studentportalen.

# Förberedelser inför nästa tillfälle

- Funktioner
  https://www.youtube.com/watch?v=lQx7qZ7XApI
- Subqueries
  https://www.youtube.com/watch?v=I4wk67fkZNw
- Stored Procedures
  http://www.mysqltutorial.org/introduction-to-sql-stored-procedures.aspx