MDE

TP1 – Class 3

CRUD + Data Queries

2024 - 2025

TOPICS

- FR Implementation (CRUD + data queries)
- Overview of Joins and Views
- Aggregation Functions

CRUD on several entities

INSERT INTO

```
insert into client(name, address, vat, telephone)
values ('Maria', 'Rua A', '123456789', '+351 91 444 77 22'),
       ('Joao',  'Rua B', '254713689', '+351 91 777 77 22'),
       ('Manuel', 'Rua C', '748312579', '+351 91 666 77 22'),
       ('Jose', 'Rua D', '347846216', '+351 91 788 77 22'),
       ('Joana', 'Rua E', '987654321', '+351 91 333 77 22'),
       ('Francisca', 'Rua f', '467342553', '+351 91 452 77 22');
```

idclient	name	address	vat	telephone
1	Maria	Rua A	123456789	+351 91 444 77 22
2	Joao	Rua B	254713689	+351 91 777 77 22
3	Manuel	Rua C	748312579	+351 91 666 77 22
4	Jose	Rua D	347846216	+351 91 788 77 22
5	Joana	Rua E	987654321	+351 91 333 77 22
6	Francisca	Rua f	467342553	+351 91 452 77 22
NULL	NULL	NULL	NULL	NULL

```
insert into installation(local_code, address, client_idclient)
values ('124578986', 'Avenida Z', 1),
       ('235147853', 'Avenida G', 1),
       ('324789142', 'Avenida W', 2),
       ('542123985', 'Avenida X', 3),
       ('235421165', 'Avenida T', 6);
```

idInstallation	local_code	address	client_idclient
1	124578986	Avenida Z	1
2	235147853	Avenida G	1
3	324789142	Avenida W	2
4	542123985	Avenida X	3
5	235421165	Avenida T	6
NULL	NULL	NULL	NULL

UPDATE

```
update client set address = 'Rua ZZ'
where idclient = 6;
```

idclient	name	address	vat	telephone
1	Maria	Rua A	123456789	+351 91 444 77 22
2	Joao	Rua B	254713689	+351 91 777 77 22
3	Manuel	Rua C	748312579	+351 91 666 77 22
4	Jose	Rua D	347846216	+351 91 788 77 22
5	Joana	Rua E	987654321	+351 91 333 77 22
6	Francisca	Rua ZZ	467342553	+351 91 452 77 22
NULL	NULL	NULL	NULL	NULL

SELECT

<to complete…>

DELETE

<to complete…>

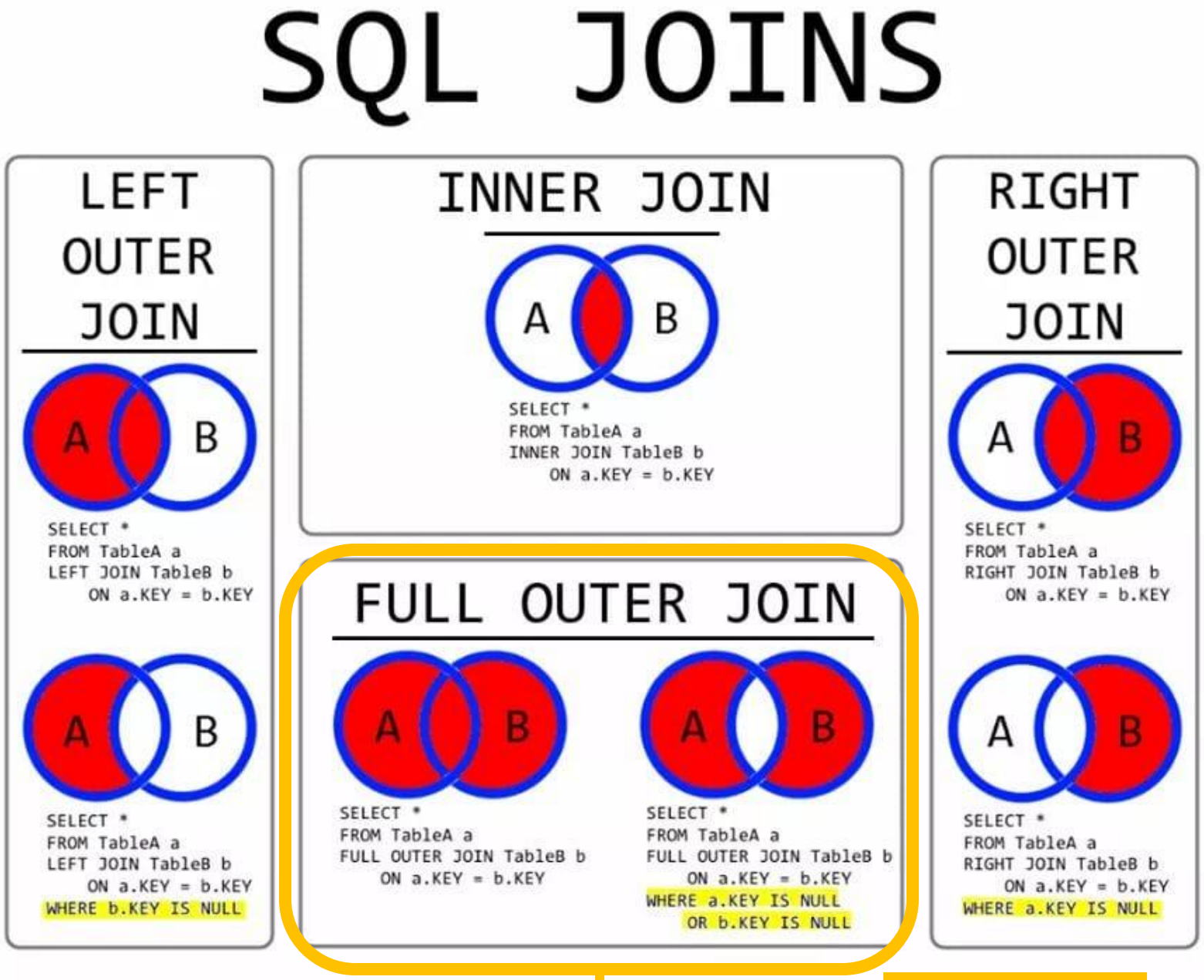
Recalling select statement (revisions)

- **Select**: to obtain data from tables.
- **Distinct**: eliminate duplicate results from the output of a query
- **Where**: specify conditions for rows in the result set returned by a query.
- **AND, OR**: combine two or more Boolean expressions and return true, used to specify conditions.
- **Order by**: sort the results
- **Fetch**: limit rows returned by a query using row limiting clause
- **IN**: determine if a value matches any value in a list or subquery.
- **Like**: perform matching based on specific patterns.
- **IS null, is not null**: check if an expression or values in a column is null or not.

JOINS

- **INNER JOIN** – show how to query rows from a table that have matching rows from another table.
- **LEFT JOIN** – select rows from the left table that have or don’t have the matching rows in the right table.
- **RIGHT JOIN** – Query rows from the right table that have or don’t have the matching rows in the left table.
- **FULL OUTER JOIN** – query data from two tables.
- **CROSS JOIN** – make a Cartesian product from multiple tables.
- **Self-join** – query hierarchical data or compare rows within the same table.

- Inclusive Left/Right/full
- Exclusive Left/Right/full (yellow CLAUSES)

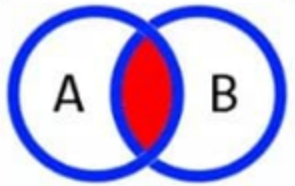


Not supported by MySQL

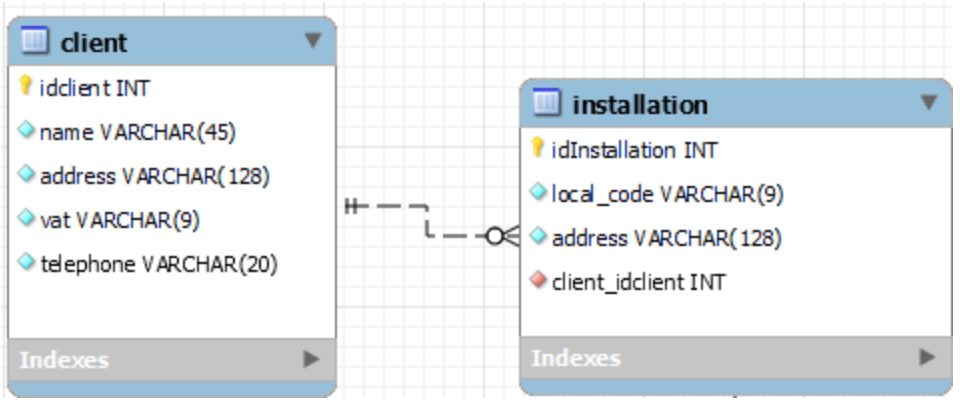
https://www.facebook.com/CollabUXWebSolutions/photos/sql-joins/1160323134121580/

Illustrating inner join

- To retrieve rows from a table that have matching rows from other tables.



- E.g., show each client and corresponding installation(s)

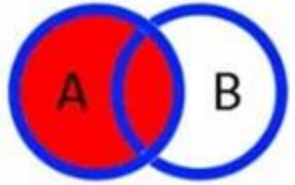



```
SELECT name, c.address, local_code as installation_code, i.address
FROM client c
INNER JOIN installation i on c.idclient = i.client_idclient
ORDER BY
    name ASC;
```

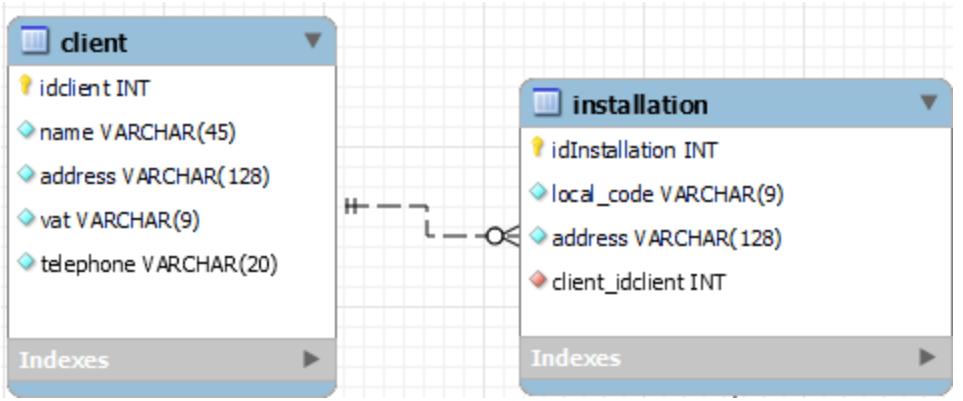
	name	address	installation_code	address
▶	Francisca	Rua ZZ	235421165	Avenida T
	Joao	Rua B	324789142	Avenida W
	Manuel	Rua C	542123985	Avenida X
	Maria	Rua A	235147853	Avenida G
	Maria	Rua A	124578986	Avenida Z

Illustrating left join

- LEFT JOIN returns all records from *A* and only those records from *B* that intersect with *A.*



- E.g., for each client show all the existing and not existing installations




```
SELECT name, c.address, local_code as installation_code, i.address
FROM client c
LEFT JOIN installation i on c.idclient = i.client_idclient
ORDER BY
    name ASC;
```

name	address	installation_code	address
Francisca	Rua ZZ	235421165	Avenida T
Joana	Rua E	NULL	NULL
Joao	Rua B	324789142	Avenida W
Jose	Rua D	NULL	NULL
Manuel	Rua C	542123985	Avenida X
Maria	Rua A	235147853	Avenida G
Maria	Rua A	124578986	Avenida Z

MySQL Views

- A view consists of a query (like the others already seen) with a name.
- So, by definition, a view is a “virtual” table whose data is the result of a stored query, which is derived each time when you query against the view.
- A view is a virtual table because it can be used like a table in SQL queries.
- Unlike a table, a view does not store any data. To be precise, a view only *behaves* like a table. And it is just a named query stored in the database.
- When you query data from a view, MySQL uses this stored query to retrieve the data from the underlying tables.

MySQL Views

- **Simplifying data retrieval**: build a complex query, test it carefully, and encapsulate the query in a view. Access the data of the underlying tables through the view instead of rewriting the whole query again and again.
- Example: installation from ‘’Maria”:

```
CREATE OR REPLACE VIEW client_installations AS
SELECT name AS client, local_code AS installation_code
FROM client c
RIGHT JOIN installation i on c.idclient = i.client_idclient
ORDER BY
    local_code ASC;

select * from client_installations where client = 'Maria';
```

	client	installation_code
▶	Maria	124578986
	Maria	235147853

Illustrating average, max, min, sum, count

```sql
-- aggregated functions
SELECT ROUND(AVG(price), 2) avg_price
FROM contract;
```

avg_price
50.82

```sql
SELECT max(price) max_price
FROM contract;
```

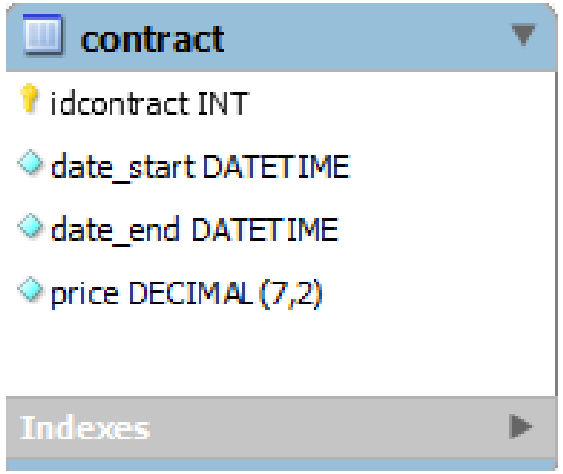
max_price
86.45

```sql
SELECT count(price) count_price
FROM contract;
```

count_price
6

```sql
SELECT SUM(price) sum_price
FROM contract
WHERE year(date_end) > 2023;
```

sum_price
91.10



idcontract	date_start	date_end	price
1	2017-11-01 ...	2022-10-01...	36.25
2	2018-08-01 ...	2023-09-01...	86.45
3	2019-09-05 ...	2024-10-01...	36.25
4	2020-09-02 ...	2025-10-01...	54.85
5	2018-09-01 ...	2023-10-02...	54.85
6	2017-09-02 ...	2022-10-03...	36.25
NULL	NULL	NULL	NULL

NEXT WEEK

- Triggers, SQL/PSM
- Finishing the implementation of the FRs