Introducion to Databases

Exercises: SQL

Schema for following examples

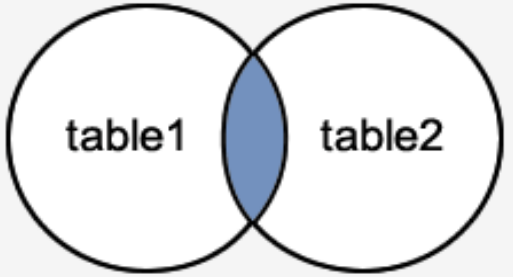
- Relation **Students**

admission_no	first_name	last_name	age	city
3354	Luisa	Evans	13	Texas
2135	Paul	Ward	15	Alaska
4321	Peter	Bennett	14	California
4213	Carlos	Patterson	17	New York
5112	Rose	Huges	16	Florida
6113	Marielia	Simmons	15	Arizona
7555	Antonio	Butler	14	New York
8345	Diego	Cox	13	California

- Relation **Fee**

admission_no	course	amount_paid
3354	Java	20000
7555	Android	22000
4321	Python	18000
8345	SQL	15000
5112	Machine Learning	30000
9272	R	10000

(INNER) JOIN



- (INNER) JOIN is used to combine more than one table to get results
- Inner join selects records that have matching values in both tables based on a clause:

```
SELECT *
FROM Student
JOIN Fee ON
Student.admission_no = Fee.admission_no;
```

admission_no	first_name	last_name	age	city	course	amount_paid
3354	Luisa	Evans	13	Texas	Java	20000
7555	Antonio	Butler	14	New York	Android	22000
4321	Peter	Bennett	14	California	Python	18000
8345	Diego	Cox	13	California	SQL	15000
5112	Rose	Huges	16	Florida	Machine Learning	30000

- We can also use WHERE instead of JOIN:

```
SELECT *
FROM Student, Fee
WHERE Student.admission_no = Fee.admission_no;
```

admission_no	first_name	last_name	age	city	course	amount_paid
3354	Luisa	Evans	13	Texas	Java	20000
7555	Antonio	Butler	14	New York	Android	22000
4321	Peter	Bennett	14	California	Python	18000
8345	Diego	Cox	13	California	SQL	15000
5112	Rose	Huges	16	Florida	Machine Learning	30000

NATURAL JOIN

- NATURAL JOIN is a type of EQUI JOIN
- Has to have at least one identically named column (of the same data type)
- Does not require the use of „ON“ clause

- Since in the previous example we were joining on admission_no, we get the same result here:

```
SELECT *
FROM Student
NATURAL JOIN Fee;
```

admission_no	first_name	last_name	age	city	course	amount_paid
3354	Luisa	Evans	13	Texas	Java	20000
7555	Antonio	Butler	14	New York	Android	22000
4321	Peter	Bennett	14	California	Python	18000
8345	Diego	Cox	13	California	SQL	15000
5112	Rose	Huges	16	Florida	Machine Learning	30000

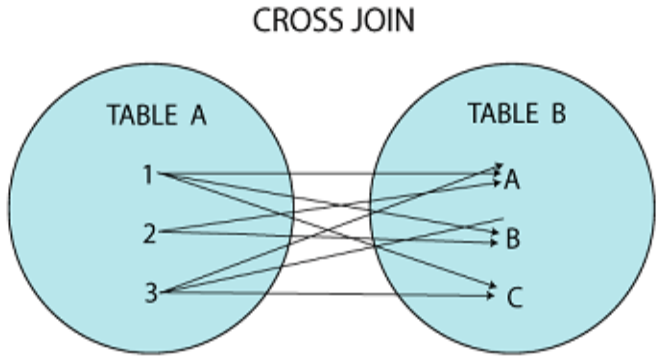
SELF JOIN

- Joins a table to itself. Each table row is combined with itself in the second table.

```
SELECT *
FROM Student S1, Student S2
WHERE S1.admission_no != S2.admission_no
AND S1.city = S2.city;
```

admission_no	first_name	last_name	age	city	admission_no	first_name	last_name	age	city
4321	Peter	Bennett	14	California	8345	Diego	Cox	13	California
4213	Carlos	Patterson	17	New York	7555	Antonio	Butler	14	New York
7555	Antonio	Butler	14	New York	4213	Carlos	Patterson	17	New York
8345	Diego	Cox	13	California	4321	Peter	Bennett	14	California

CROSS JOIN



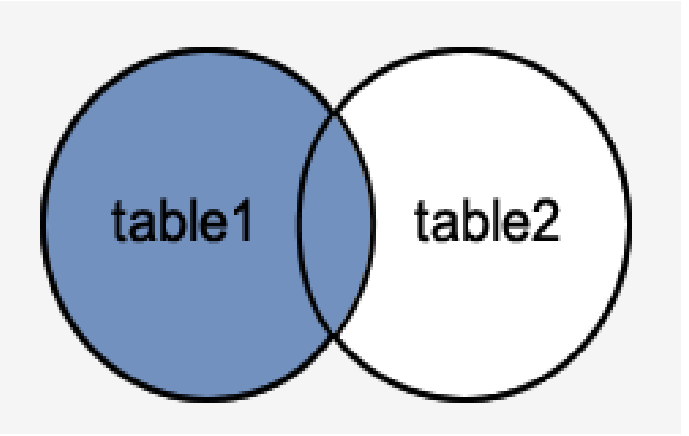
- Cross join is the same as cartesian product:

```
SELECT *
FROM Student
CROSS JOIN Fee
WHERE Student.admission_no = Fee.admission_no;
```

admission_no	first_name	last_name	age	city	admission_no	course	amount_paid
3354	Luisa	Evans	13	Texas	3354	Java	20000
7555	Antonio	Butler	14	New York	7555	Android	22000
4321	Peter	Bennett	14	California	4321	Python	18000
8345	Diego	Cox	13	California	8345	SQL	15000
5112	Rose	Huges	16	Florida	5112	Machine Learning	30000

- If we do not use the WHERE clause, we get a typical cartesian product (everything by everything)

LEFT (OUTER) JOIN

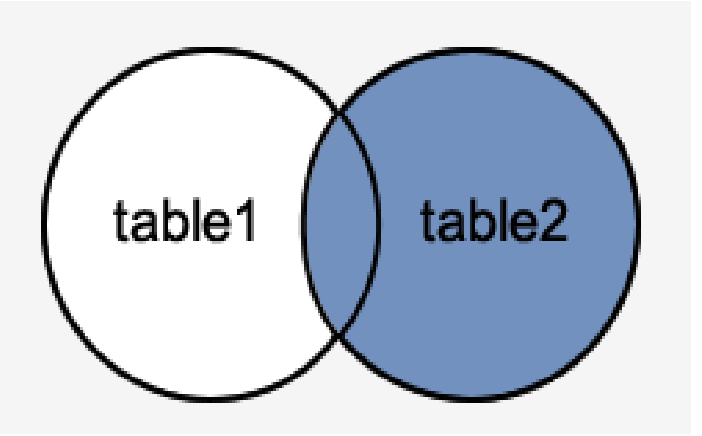


- The left (outer) join returns all records from the left table and the matchings with the records from the right table

```
SELECT *
FROM Student
LEFT JOIN Fee
ON Student.admission_no = Fee.admission_no;
```

admission_no	first_name	last_name	age	city	admission_no	course	amount_paid
3354	Luisa	Evans	13	Texas	3354	Java	20000
7555	Antonio	Butler	14	New York	7555	Android	22000
4321	Peter	Bennett	14	California	4321	Python	18000
8345	Diego	Cox	13	California	8345	SQL	15000
5112	Rose	Huges	16	Florida	5112	Machine Learning	30000
4213	Carlos	Patterson	17	New York	*NULL*	*NULL*	*NULL*
2135	Paul	Ward	15	Alaska	*NULL*	*NULL*	*NULL*
6113	Marielia	Simmons	15	Arizona	*NULL*	*NULL*	*NULL*

RIGHT (OUTER) JOIN

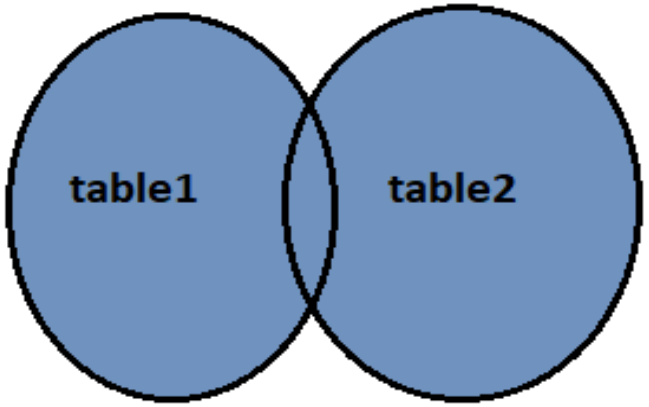


- The right (outer) join returns all records from the right table and the matchings with the records from the left table

```
SELECT *
FROM Student
RIGHT JOIN Fee
ON Student.admission_no = Fee.admission_no;
```

admission_no	first_name	last_name	age	city	admission_no	course	amount_paid
3354	Luisa	Evans	13	Texas	3354	Java	20000
7555	Antonio	Butler	14	New York	7555	Android	22000
4321	Peter	Bennett	14	California	4321	Python	18000
8345	Diego	Cox	13	California	8345	SQL	15000
5112	Rose	Huges	16	Florida	5112	Machine Learning	30000
NULL	*NULL*	*NULL*	*NULL*	*NULL*	9272	R	10000

FULL OUTER JOIN



- The full (outer) join returns all records when there is a match in the left or the right table

```
SELECT *
FROM Student
FULL JOIN Fee
ON Student.admission_no = Fee.admission_no;
```

admission_no	first_name	last_name	age	city	admission_no	course	amount_paid
3354	Luisa	Evans	13	Texas	3354	Java	20000
7555	Antonio	Butler	14	New York	7555	Android	22000
4321	Peter	Bennett	14	California	4321	Python	18000
8345	Diego	Cox	13	California	8345	SQL	15000
5112	Rose	Huges	16	Florida	5112	Machine Learning	30000
NULL	*NULL*	*NULL*	*NULL*	*NULL*	9272	R	10000
4213	Carlos	Patterson	17	New York	*NULL*	*NULL*	*NULL*
2135	Paul	Ward	15	Alaska	*NULL*	*NULL*	*NULL*
6113	Marielia	Simmons	15	Arizona	*NULL*	*NULL*	*NULL*

Operators over a set

- These operators are *binary* - they normally take two parameters:

```
=     equals
>     greater than
<     less than
>=    greater or equal
<=    less or equal
```

- You can use the words **ALL** or **ANY** where the right side of the operator might have multiple values.
- Show each students name and age that is older than ALL students from California. (Note that we mean older than every single enrolled Californian not the combined age)

```
SELECT first_name, age
FROM student
WHERE age > ALL ( SELECT age
                  FROM student
                  WHERE city='California');
```

first_name	age
Paul	15
Carlos	17
Rose	16
Marielia	15

Nested SELECT

- The **result** of a SELECT statement may be used **as a value** in another statement.
- Let's say we want to list only courses that have an above average amount of fee.
  - (Here we calculated average by hand and compared it)

```
SELECT course, amount_paid
FROM fee
WHERE amount_paid > 19166;
```

course	amount_paid
Java	20000
Android	22000
Machine Learning	30000

- Of course more elegant solution is to calculate average „on the fly“ using nested select

```
SELECT course, amount_paid
FROM fee
WHERE amount_paid > (
       SELECT AVG(amount_paid)
       FROM fee);
```

course	amount_paid
Java	20000
Android	22000
Machine Learning	30000

CASE

- CASE allows you to **return different values under different conditions**.
- If there no conditions match (and there is no ELSE) then NULL is returned.

```
SELECT course, amount_paid,
CASE
WHEN amount_paid > 20000 THEN 'Expensive'
WHEN amount_paid = 20000 THEN 'Reasonable'
ELSE 'Not expensive'
END AS price_range
FROM fee;
```

course	amount_paid	price_range
Java	20000	Reasonable
Android	22000	Expensive
Python	18000	Not expensive
SQL	15000	Not expensive
Machine Learning	30000	Expensive
R	10000	Not expensive

GROUP BY

- GROUP BY statement groups rows that have the same values into summary rows
- GROUP BY is often used with aggregate functions (COUNT, SUM, AVG MAX, MIN)
- ORDER BY orders the result based on the variable provided (can be ASC or DESC)

```
SELECT SUM(age), city
FROM Student
GROUP BY city
ORDER BY SUM(age) DESC;
```

sum	city
31	New York
27	California
16	Florida
15	Alaska
15	Arizona
13	Texas

If you want to run the code yourself (1)

CREATE TABLE Student(admission_no varchar(45) NOT NULL, first_name varchar(45) NOT NULL, last_name varchar(45) NOT NULL, age int, city varchar(25) NOT NULL);	Creates table Student
INSERT INTO Student (admission_no, first_name, last_name, age, city) VALUES (3354,'Luisa', 'Evans', 13, 'Texas‘), (2135, 'Paul', 'Ward', 15, 'Alaska‘), (4321, 'Peter', 'Bennett', 14, 'California'), (4213,'Carlos', 'Patterson', 17, 'New York'), (5112, 'Rose', 'Huges', 16, 'Florida‘), (6113, 'Marielia', 'Simmons', 15, 'Arizona'), (7555,'Antonio', 'Butler', 14, 'New York‘), (8345, 'Diego', 'Cox', 13, 'California');	Inserts values into Student

If you want to run the code yourself (2)

CREATE TABLE Fee(**admission_no varchar(45) NOT NULL,** **course varchar(45) NOT NULL,** **amount_paid int** **);**	**Creates table Fee**
INSERT INTO Fee (admission_no, course, amount_paid) **VALUES (3354,'Java', 20000),** **(7555, 'Android', 22000),** **(4321, 'Python', 18000),** **(8345,'SQL', 15000),** **(5112, 'Machine Learning', 30000),** **(9272, 'R', 10000);**	**Inserts values into Fee**