**Практична робота №7. Поєднання таблиць**

**Мета:** Вивчення особливостей поєднання таблиць, які відіграють одну з ключових ролей в видобуванні інформації з БД Набуття практичних навичок поєднання таблиць за допомогою SQL-запитів.

***Методичні рекомендації***

*При вивченні теми слід звернути увагу на створення та збереження структури бази даних(БД), редагування структури, визначення типів полів, зв’язування таблиць у БД.*

1. Повторити лекції №7, 8, теоретичну частину ЛР № 4-8 та ПР№6. Ознайомитися з теоретичною частиною цієї ЛР.
2. За результатами виконання ЛР № 4,5,6 та ПР№6 повинна бути сформована та заповнена реляційна БД з таблицями Викладач, Студент, Групи, Предмети, Розклад, Аудиторії, Предмети, Оцінки. Для цих таблиць виконати завдання:

* Сформулювати запит на виконання внутрішнього об’єднання
* Сформулювати запит на виконання зовнішнього об’єднання
* Побудувати комбіновані запити UNION

Таблиці та функції обрати самостійно.

1. Результати оформити звітом, в якому надаються сформовані SQL-запити, опис дій, що вони виконують, та скріншот виконання на вашій БД.
2. Результати надсилати на електронну адресу викладача [t.i.lumpova@gmail.com](mailto:t.i.lumpova@gmail.com)

Файл повинен мати назву в такому форматі:

**DB<Номер групи><Номер лекції / практичної / лабораторної [літера позначення типу роботи L – лекція, P – практична, R – лабораторна]<Прізвище англійською>**. Наприклад, **DB3101R**buts.doc.

Не копіюйте фрагментів з різних інформаційних джерел, подумайте і викладіть свою точку зору. При наявності робіт -"близнюків" відповідь буде зараховуватися першому за часом надсилання.

**Строк виконання цієї роботи ІПЗ-31 - 19.11.2023**

**ІПЗ-32 - 18.11.2023**

***Контрольні запитання***

1. Для чого на Вашу думку використовується об’єднання таблиць?
2. За допомогою яких SQL-операторів здійснюється внутрішнє об’єднання?
3. З якою метою використовуються псевдоніми таблиць?
4. В чому полягає природне об’єднання таблиць?
5. В чому полягає зовнішнє об’єднання таблиць?
6. Для чого використовується оператор UNION?
7. Які правила при написанні комбінованих запитів UNION?
8. Як речення ORDER BY використовується в комбінованих запитах UNION?

**МЕТОДИЧНІ ВКАЗІВКИ**

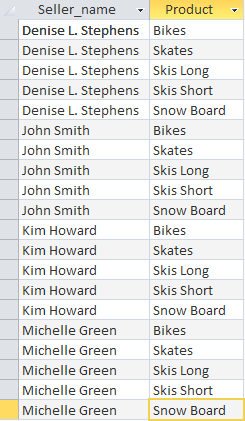
## *Поєднання таблиць (INNER JOIN*)

Найбільш потужною особливістю мови SQL є можливість поєднувати різні таблиці в оперативній пам'яті СУБД під час виконання запитів. Об'єднання дуже часто використовуються для аналізу даних. Як правило, дані знаходяться в різних таблицях, що дозволяє їх більш ефективно зберігати (оскільки інформація не дублюється), спрощує обробку даних та дозволяє масштабувати базу даних (можливо додавати нові таблиці з додатковою інформацією). Таблиці баз даних, які використовуються в СУБД Access являються реляційними таблицями, тобто всі таблиці можна пов'язати між собою по спільним полям.

**1. Створення об'єднання таблиць**

Об'єднання таблиць дуже проста процедура. Потрібно вказати всі таблиці, які будуть включені в об'єднання та "пояснити" СУБД, як вони будуть пов'язані між собою. Поєднання робиться за допомогою слова **WHERE**, наприклад:

**SELECT DISTINCT Seller\_name, Product FROM Sellers, Sumproduct WHERE Sellers.City = Sumproduct.City**

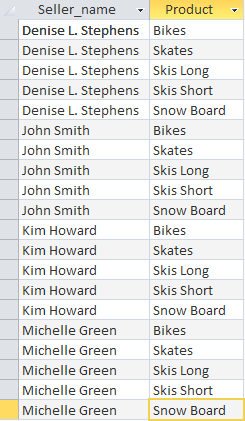


Поєднавши дві таблиці, ми змогли побачити які товари реалізує кожен продавець. Розглянемо код запиту детальніше, оскільки він трохи відрізняється від звичайного запиту. Оператор **SELECT** починається з вказання стовпців, які ми хочемо вивести, проте ці поля знаходяться в різних таблицях, речення **FROM** містить дві таблиці, які ми хочемо поєднати в операторі **SELECT**, таблиці поєднуються за допомогою слова **WHERE**, яке вказує стовпці для об'єднання. Обов'язково потрібно вказувати повну назву поля (*Таблиця.Поле*), оскільки поле **City** є в обох таблицях.

**2. Внутрішнє об'єднання**

В попередньому прикладі для об'єднання таблиць ми використали слово **WHERE**, яке здійснює перевірку на основі еквівалентності двох таблиць. Об'єднання такого типу називається також "*внутрішнім об'єднанням*". Існує також і інший спосіб об'єднання таблиць, який явно вказує на тип об'єднання. Розглянемо наступний приклад:

**SELECT DISTINCT Seller\_name, Product FROM Sellers INNER JOIN Sumproduct ON Sellers.City = Sumproduct.City**



В цьому запиті замість **WHERE** ми використали конструкцію **INNER JOIN ... ON ...**, яка дала аналогічний результат. Незважаючи на те, що об'єднання з реченням **WHERE** є коротшим, все таки краще використовувати **INNER JOIN**, оскільки вона є більш гнучкою.

# *Розширене поєднання таблиць (OUTER JOIN)*

Ми розглянули самі прості способи об'єднання таблиць - за допомогою речень **WHERE** та **INNER JOIN**. Ці об'єднання називаються внутрішніми об'єднаннями або об'єднаннями по еквівалентності. Проте SQL має у своєму арсеналі набагато більше можливостей поєднати таблиці, а саме існують також й інші види об'єднань: зовнішні об'єднання, природні об'єднання та самооб'єднання. Розглянемо, яким чином ми можемо присвоювати таблицям псевдоніми, оскільки, в подальшому, ми будемо змушені використовувати повні назви полів (Таблиця.Поле), якими без скорочень буде дуже важко оперувати через їх велику довжину.

## 1. Використання псевдонімів таблиць

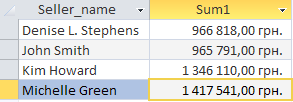
В попередньому розділі ми дізналися, як можна використовувати псевдоніми для посилання на певні поля таблиці або на розрахункові поля. SQL так само надає нам можливість використовувати псевдоніми замість імен таблиць. Це надає нам такі переваги, як більш короткий синтаксис SQL та дозволяє багато раз використовувати одну й ту ж таблицю в операторі **SELECT**.

**SELECT Seller\_name, SUM(Amount) AS Sum1**

**FROM Sellers AS S, Sumproduct AS SP**

**WHERE S.City = SP.City**

**GROUP BY Seller\_name**



Ми відобразили загальну суму реалізованого товару по кожному продавцю. В нашому SQL запиті ми використали такі псевдоніми: для розрахункового поля **SUM(Amount)** псевдонім **Sum1**, для таблиці **Sellers** псевдонім **S** та для **Sumproduct** псевдонім **SP**. Зауважимо, що псевдоніми таблиць можуть бути застосовані також і в інших реченнях, як **ORDER BY**, **GROUP BY** та інших.

## 2. Самооб'єднання

Розглянемо приклад. Припустимо, нам потрібно дізнатися адресу продавців, які торгують в тій самій країні, що і John Smith. Для цього створимо такий запит:

**SELECT City, Country, Seller\_name**

**FROM Sellers**

**WHERE Country = (SELECT Country FROM Sellers WHERE Seller\_name = 'John Smith')**

http://moonexcel.com.ua/images/sql11_ua_2.png

Також, цю задачу ми можемо вирішити і через самооб'єднання, прописавши наступний код:

**SELECT S1.Address, S1.City, S1.Country, S1.Seller\_name**

**FROM Sellers AS S1, Sellers AS S2**

**WHERE S1.Country = S2.Country AND S2.Seller\_name = 'John Smith'**

http://moonexcel.com.ua/images/sql11_ua_3.png

Для вирішення цієї задачі використовувалися псевдоніми. Перший раз для таблиці **Sellers** присвоїли псевдонім **S1**, другий раз - псевдонім **S2**. Після цього ці псевдоніми можна застосовувати в якості імен таблиць. В операторі **WHERE** ми до назви кожного поля додаємо префікс **S1**, для того, щоб СУБД розуміла поля якої таблиці потрібно виводити (оскільки ми з однієї таблиці зробили дві віртуальні). Речення **WHERE** спочатку об'єднує таблиці, а потім фільтрує дані другої таблиці по полю **Seller\_name**, щоби повернути лише необхідні значення.

Самооб'єднання часто використовують для заміни підзапитів, які вибирають дані з тієї ж таблиці, що і зовнішній оператор **SELECT**. Хоча кінцевий результат виходить тим самим, багато СУБД опрацьовують об'єднання набагато швидше ніж підзапити. Потрібно проекспериментувати, щоби визначити, який запит працює швидше.

## 3. Природне об'єднання

Природне об'єднання - це об'єднання, в якому ви вибираєте тільки ті стовпці, які не повторюються. Зазвичай це робиться за допомогою запису (**SELECT \***) для однієї таблиці і вказівкою переліку полів - для решти таблиць. Наприклад:

**SELECT SP.\*, S.Country**

**FROM Sumproduct AS SP, Sellers AS S**

**WHERE SP.City = S.City**

В цьому прикладі метасимвол (\*) використовується тільки для першої таблиці. Всі решта стовпці вказані явно, тому дублікати стовпців не вибираються.

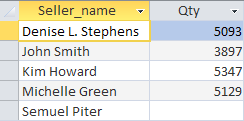
## 4. Зовнішнє об'єднання (OUTER JOIN)

Зазвичай, при об'єднанні зв'язують рядки однієї таблиці з відповідними рядками іншої, проте в деяких випадках може знадобитися включати в результат рядки, які не мають пов'язаних рядків в іншій таблиці (тобто вибираються геть усі рядки з однієї таблиці та додаються лише пов'язані рядки з іншої). Об'єднання такого типу називається зовнішнім. Для цього використовуються ключові слова **OUTER JOIN ... ON ...** з приставкою **LEFT** або **RIGHT**. Розглянемо приклад, попередньо добавивши в таблицю **Sellers** нового продавця - Semuel Piter, який ще немає продаж:

**SELECT Seller\_name, SUM(Quantity) AS Qty**

**FROM Sellers LEFT OUTER JOIN Sumproduct ON Sellers.City=Sumproduct.City**

**GROUP BY Seller\_name**



Даним запитом ми витягнули перелік усіх продавців в базі та підрахували для них загальну кількість проданого товару за усі місяці. Бачимо що по новому продавцю Semuel Piter відсутні продажі. Якби ми використали внутрішнє об'єднання, то нового продавця ми б не побачили, оскільки він немає записів в таблиці **Sumproduct**. Ми також можемо змінювати напрямок поєднання не лише прописуючи **LEFT** або **RIGHT**, але й просто змінюючи порядок таблиць (тобто такі два записи будуть давати однаковий результат: **Sellers LEFT OUTER JOIN Sumproduct** та **Sumproduct RIGHT OUTER JOIN Sellers**).

Також деякі СУБД дозволяють здійснювати зовнішнє об'єднання за спрощеним записом, використовуючи знаки **\*=** та **=\***, що відповідає **LEFT OUTER JOIN** та **RIGHT OUTER JOIN** відповідно. Таким чином попередній запит можна було б переписати так:

**SELECT Seller\_name, SUM(Quantity) AS Qty**

**FROM Sellers, Sumproduct**

**WHERE Sellers.City \*= Sumproduct.City**

Нажаль **Access** не підтримує скорочений запис для зовнішнього об'єднання.

## 5. Повне зовнішнє об'єднання (FULL OUTER JOIN)

Також існує й інший тип зовнішнього об'єднання - повне зовнішнє об'єднання, яке відображає усі рядки з обох таблиць та пов'язує лише ті, які можуть бути пов'язані. Синтаксис повного зовнішнього об'єднання наступний:

**SELECT Seller\_name, Product**

**FROM Sellers FULL OUTER JOIN Sumproduct ON Sellers.City=Sumproduct.City**

Знову ж, повне зовнішнє об'єднання не підтримують такі СУБД: **Access, MySQL, SQL Server та Sybase**.

# *Комбіновані SQL-запити (UNION*)

В більшості **SQL-запитів** використовується один оператор, за допомогою якого повертаються дані із однієї або кількох таблиць. **SQL** також дозволяє виконувати одночасно кілька окремих запитів та відображати результат у вигляді єдиного набору даних. Такі комбіновані запити зазвичай називають *поєднаннями* або *складними запитами*.

**1. Використання оператора UNION**

Запити в мові **SQL** комбінуються за допомогою оператора **UNION**. Для цього необхідно вказати кожен запит **SELECT** та розмістити між ними ключове слово **UNION**. Обмежень щодо кількості використань оператора **UNION** в одному загальному запиті немає.

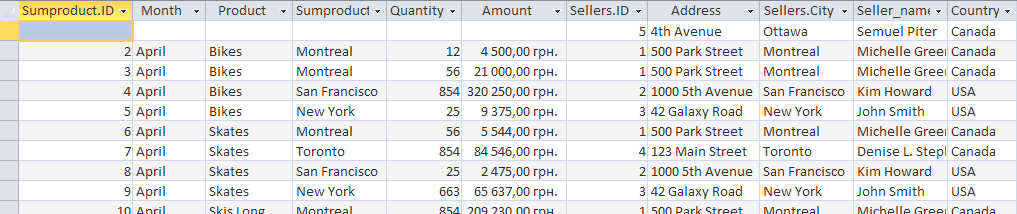
**Access** не має можливості створювати *повне зовнішнє об'єднання*, тепер ми подивимось, як можна цього досягнути через оператор **UNION**.

**SELECT \***

**FROM Sumproduct LEFT JOIN Sellers ON Sumproduct.City = Sellers.City**

**UNION  
SELECT \***

**FROM Sumproduct RIGHT JOIN Sellers ON Sumproduct.City = Sellers.City**

****

Бачимо, що запит відобразив як всі колонки з першої таблиці - так і з другої, незалежно від того, чи всі записи мають відповідники у іншій таблиці.

В багатьох випадках замість **UNION** ми можемо використовувати речення **WHERE** з багатьма умовами, та отримувати аналогічний результат. Проте через **UNION** записи виглядають більш лаконічними та зрозумілими. Також необхідно дотримуватись певних правил при написанні комбінованих запитів:

* запит **UNION** повинен включати два і більше операторів **SELECT**, відділених між собою ключовим словом **UNION** (тобто якщо в запиті використовується чотири оператора SELECT, то повинно бути три ключових слова UNION);
* кожен запит в операторі **UNION** повинен мати одні й ті ж стовпці, вирази чи статистичні функції, які, до того ж, мають бути перераховані в однаковому порядку;
* типи даних стовпців мають бути сумісними. Вони не обов'язково мають бути одного типу, проте мають мати подібний тип, щоби **СУБД** могла їх однозначно перетворити (наприклад, це можуть бути різні числові типи даних або різні типи дати).

**2. Включення або виключення повторюваних рядків**

Запит з **UNION** автоматично видаляє усі повторювані рядки з набору результатів запиту (іншими словами, веде себе як речення **WHERE** з кількома умовами в одному операторі **SELECT**). Така поведінка оператора **UNION** по замовчуванню, але при бажанні ми можемо змінити це. Для цього нам варто використовувати оператор **UNION ALL** замість **UNION**.

**3. Сортування результатів комбінованих запитів**

Результати виконання оператора **SELECT** сортуються за допомогою речення **ORDER BY**. При комбінуванні запитів за допомогою **UNION** тільки одне речення **ORDER BY** може бути використане, і воно має бути проставлене в останньому операторі **SELECT**. Дійсно, на практиці немає особливого змісту частину результатів сортувати в одному порядку, а іншу частину - в іншому. Тому кілька речень **ORDER BY** застосовувати не дозволяється.

### *Об'єднання двох і більш SQL запитів SELECT в базах даних SQLite*

SQL запити SELECT бувають громіздкими не тільки через те, що використовують відразу всі уточнюючі фрази, а й з-за того, чтоSQLite3 дає можливість об'єднувати SQL запити SELECT, вірніше буде сказати об'єднувати результати двох запитів SELECT за допомогою ключового слова UNION. Найчастіше використання UNION і SELECT можуть дати дуже цікаві результати. Ми можемо написати SQL запит SELECT використовуючи всі уточнюючі фрази. потім написати UNION і написати ще один десятиповерховий SELECT, і SQLite виконає такий запит, а у вас в результаті буде одна таблиця значення в якій будуть з двох таблиць: спочатку будуть виведені рядки з першої таблиці, а потім до цих рядків будуть додані рядки з другої таблиці. Можна об'єднувати три і більше запитів

### *Порівняння результатів двох SQL запитів SELECT в базах даних SQLite*

SQL запит SELECT завжди повертає нам таблицю, це дуже важливо для розуміння його роботи. Тому що SQL запит SELECT не тільки робить вибірку даних, але і дозволяє порівнювати результати вибірки даних за допомогою ключових слів EXCEPT і INTERSECT. Перша ключова фраза дає можливість отримати тільки унікальні рядки з двох або більше проміжних таблиць, а друга дозволяє записати в результуючу таблицю тільки повторювані рядки з двох таблиць, отриманих в підзапитах.

Коли ви будете використовувати SQL запит SELECT, ви неминуче зіткнетеся з тим, що SQLite (на місці SQLite може бути будь-яка інша СУБД) буде порівнювати значення в таблицях або порівнювати рядки. Тому, по-перше, вам потрібно розуміти, як SQLite порівнює значення. А по-друге, знати про типи даних в SQLite:

### *Використання SQL операторів разом з SELECT в базах даних SQLite*

Для ускладнення логіки запитів на вибірку даних ми можемо використовувати різні SQL оператори. про них ми говорили раніше і перераховували їх. SELECT може використовувати логічні вираження AND і OR, щоб зробити умови вибірки більш точними. У SQL запитах SELECT ми можемо використовувати оператор BETWEEN. Оператор BETWEEN і SELECT дозволяють отримати значення в діапазоні, який як раз-таки і задається оператором BETWEEN. Ще у нас є можливість використовувати оператор LIKE. Оператор LIKE і SELECT дозволяють зробити пошук за шаблоном або ще можна сказати, що за допомогою комбінації LIKE ми можемо здійснити порівняння рядків із заданим шаблоном. А результатом вибірки буде таблиця, яка містить тільки ті рядки, в яких є підрядок, зазначена в шаблоні LIKE.

Іноді буває потрібно, щоб в результуючій таблиці, отриманої за допомогою SQL запиту SELECT. були рядки, що містять в своїх шпальтах тільки певний набір значень, для того, щоб зробити таку вибірку нам потрібно використовувати оператор IN. Результуюча таблиця, отримана після комбінації оператор IN і SELECT буде містити рядки з певного набору значень.

### *Підзапити SELECT в базах даних SQLite*

Підзапити SELECT реалізуються досить просто: ми пишемо основний або зовнішній запит SELECT, далі ставимо умову (якщо ми задаємо умову, то неминуче використовуємо SQL-оператор), але в якості правого операнда ми задаємо не якесь конкретне значення, а пишемо запит SELECT в дужках.

Підзапити отримали таку назву від того, що для їх реалізації використовується дві або більше команди SELECT, виходить так, що, наприклад, один SELECT буде вкладений в інший. Перша команда SELECT називается зовнішнім запитом, а другий SQL запит SELECT-внутрішнім підзапитом.

Найчастіше підзапити SELECT використовуються з оператором EXISTS або з оператором IN. Якщо ви пишете підзапит з використанням EXISTS, то ви побачите результат тільки в тому випадку, коли правий операнд (власне, підзапит або внутрішній запит) буде мати значення TRUE, до речі, якщо підзапит поверне NULL, то зовнішній запит буде виконаний.

Іншими словами: якщо підзапит поверне значення FALSE, то зовнішній запит навіть не буде виконуватися.

### *Об'єднання двох таблиць SQL запитом SELECT в базах даних SQLite*

Найприємніша частина SQL запиту SELECT. але в той же час і найскладніша, хоча перед тим як її швидше надумана, ніж дійсна. SQL запити SELECT дозволяють об'єднувати таблиці бази даних. Об'єднання таблиць в SQL здійснюється за допомогою ключового слова JOIN. Стандарт SQL виділяє кілька видів об'єднання таблиць JOIN:

1. Внутрішнє об'єднання: INNER JOIN.
2. Повне об'єднання: FULL JOIN.
3. Праве зовнішнє об'єднання: RIGHT OUTER JOIN.
4. Ліве зовнішнє об'єднання: LEFT OUTER JOIN.
5. Перехресне об'єднання: CROSS JOIN.

Для зовнішнього об'єднання можна опускати ключове слово OUTER, принаймні SQLite це дозволяє зробити. Але, є один мінус у SQLite, в даній СУБД реалізовано тільки три об'єднання: LEFT, CROSS і INNER. Ось таке ось обмеження творчості при написанні SQL запитів SELECT ввели розробники SQLite3.

### *Використання SQL запита SELECT з іншими командами SQLite*

SQL запит SELECT може бути використаний з іншими командами мови SQL. Природно, ми можемо виконувати запити SELECT всередині транзакцій, тому команди управління транзакціями навіть не обговорюються. У SQLite3 немає команд визначення доступу до даних.

1. Ми можемо створювати таблиці в базі даних за допомогою команди CREATE TABLE. але також SQLite дає нам можливість створювати таблиці з використанням ключового слова AS і SQL запиту SELECT.
2. Ми не можемо використовувати SQL запит SELECT, коли хочемо видалити таблицю з бази даних за допомогою команди DROP.
3. Також ми не можемо використовувати запит SELECT, якщо ми хочемо модифікувати таблицю або інший об'єкт бази даних за допомогою команди ALTER.

Останніх два пункти обумовлені тим, що перераховані вище команди працюють на рівні об'єктів, а SELECT працює з даними.

Команди маніпуляції даними:

1. Коли ми додаємо рядки в таблицю бази даних за допомогою команди INSERT. то можемо це робити за допомогою підзапиту.
2. Якщо у нас є необхідність видалити рядки з таблиці бази даних. ми використовуємо команду DELETE не так часто, але все ж, ми можемо вибрати рядки для видалення за допомогою SQL запиту SELECT.
3. Для модифікації даних в базі даних використовується команда UPDATE. Щоб вона працювала, ми повинні вибрати рядки, які необхідно модифікувати. А що може бути краще для вибору рядків, ніж SQL запит SELECT?

### *Прискорення SQL запитів SELECT в реляційних базах даних*

Ми можемо оптимізувати роботу команди SELECT в SQLite і інших реляційних СУБД. Прискорення вибірки даних відбувається за рахунок створення індексів в таблиці бази даних. Індекси дуже сильно прискорюють вибірку даних, але в замін вони роблять інші операції маніпуляції даними більш дорогими.

SQLite створює внутрішній індекс для кожної таблиці (стовпець ROWID), який може збігатися з індексом, створеним за допомогою обмеження первинного ключа PRIMARY KEY (трохи теорії про ключі і ключові атрибути). Так само ми можемо організовувати зв'язок між таблицями за допомогою обмеження зовнішнього ключа FOREIGN KEY. такий підхід не тільки забезпечує цілісність даних в базі даних. але і прискорює вибірку даних. Це ще одне застосування обмежень рівня таблиці в SQL і реляційних базах даних.

**Приклади.**

З’єднання таблиці з собою

Знайти усі пари замовників, які мають однаковий рейтинг.

Рішення 1 (з дублюванням):

SELECT c1.cname, c2.cname, c1.rating

FROM Customers c1 JOIN Customers c2

ON c1.rating = c2.rating;

Рішення 2 (без дублювання):

SELECT c1.cname, c2.cname, c1.rating

FROM Customers c1, Customers c2

WHERE c1.rating = c2.rating

AND c1.cname < c2.cname;

Або

SELECT c1.cname, c2.cname, c1.rating

FROM Customers c1 JOIN Customers c2

ON c1.rating = c2.rating

AND c1.cname < c2.cname;

**Робота на ЛР**

EXECUTING ALL IN 'SQL 1'

--

-- At line 1:

SELECT St.name, G.Sub\_code, G.grade, avg(G.grade)

FROM Grades G,Students St

WHERE G.CodeStud=St.ID\_stud --and G.Sub\_code<10

--G.CodeStud in (SELECT ID\_stud from Students where Id\_Gr =17) and

--G.CodeStud in (select ID\_stud from Students WHERE Name2 is null)

group by G.Sub\_code,G.CodeStud

-- Result: 25 rows returned in 69ms

-- EXECUTING ALL IN 'SQL 1'

--

-- At line 1:

SELECT St.name, G.Sub\_code, G.grade, round(avg(G.grade),1) AVG\_G

FROM Grades G,Students St

WHERE G.CodeStud=St.ID\_stud --and G.Sub\_code<10

--G.CodeStud in (SELECT ID\_stud from Students where Id\_Gr =17) and

--G.CodeStud in (select ID\_stud from Students WHERE Name2 is null)

group by G.Sub\_code,G.CodeStud

-- Result: 25 rows returned in 38ms

-- EXECUTING ALL IN 'SQL 1'

--

-- At line 1:

SELECT St.name, G.Sub\_code, G.grade, round(avg(G.grade),1) AVG\_G

FROM Students St LEFT OUTER JOIN Grades G

--WHERE

ON G.CodeStud=St.ID\_stud --and G.Sub\_code<10

--G.CodeStud in (SELECT ID\_stud from Students where Id\_Gr =17) and

--G.CodeStud in (select ID\_stud from Students WHERE Name2 is null)

group by G.Sub\_code,G.CodeStud

-- Result: 25 rows returned in 23ms

SELECT St.name Name, G.Sub\_code, G.grade, round(avg(G.grade),1) AVG

FROM Students St LEFT OUTER JOIN Grades G

ON G.CodeStud=St.ID\_stud --and G.Sub\_code<10

group by G.Sub\_code,G.CodeStud

ORDER by Name

SELECT St.name, G.Name\_gr

FROM Groups G LEFT OUTER JOIN Students St

--WHERE

ON G.Id\_Gr=St.Id\_Gr

SELECT St.name, G.Name\_gr

FROM Students St LEFT OUTER JOIN Groups G

--WHERE

ON G.Id\_Gr=St.Id\_Gr

SELECT St.name, G.Name\_gr

FROM Students St INNER JOIN Groups G

--WHERE

ON G.Id\_Gr=St.Id\_Gr

SELECT St.name, G.Name\_gr

FROM Groups G INNER JOIN Students St

--WHERE

ON G.Id\_Gr=St.Id\_Gr

SELECT St.name, G.Name\_gr

FROM Groups G CROSS JOIN Students St

--WHERE

ON G.Id\_Gr=St.Id\_Gr

SELECT St.name, G.Name\_gr

FROM Students St CROSS JOIN Groups G

--WHERE

ON G.Id\_Gr=St.Id\_Gr

SELECT St.ID\_stud ID, St.name NAME FROM Students St

UNION

SELECT ID\_T,Name\_T FROM Teachers

ORDER by Name

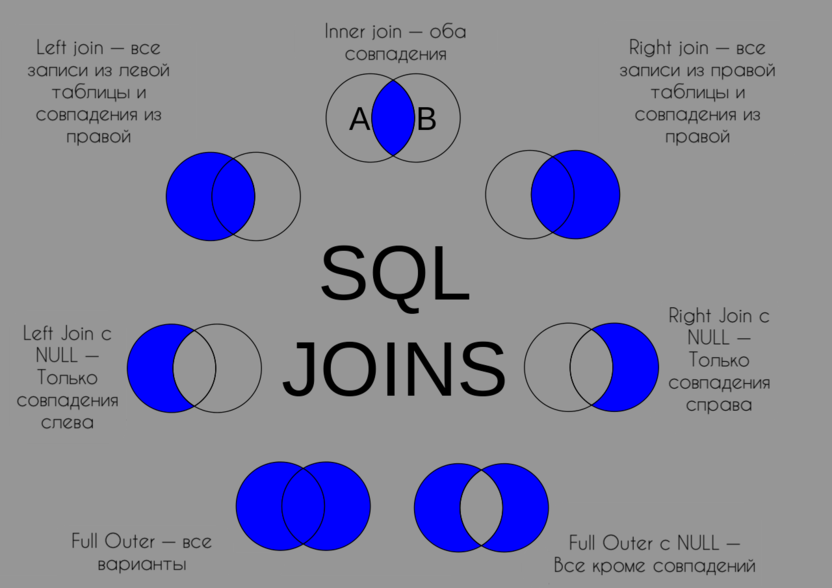
**Додатково за результатами обговорення**

**Cross Join** або перехресне з’єднання створює набір рядків, де кожен рядок з однієї таблиці з’єднується з кожним рядком з другої таблиці.

**В чому різниця між cross join та natural join**? **cross join** створює **перехресне** з‘єднання або декартовий добуток двох таблиць, тоді як **natural join** будується на всіх стовпцях, які мають одинакові імена та типи даних в обох таблицях.

**В чому різниця** cross JOIN та full JOIN?

FULL OUTER JOIN , не рядки, які не співпадають, з обох таблиць повертаються в додаток до рядків, які співпадають. **CROSS JOIN , створює декартовий добуток таблиць цілком, ключі з‘єднання не вказуються**.



Додамо студента з не заповненим Id\_Gr

SELECT St.name, G.Name\_gr

FROM Groups G LEFT JOIN Students St

ON G.Id\_Gr=St.Id\_Gr WHERE St.Id\_Gr is null

SELECT St.name, G.Name\_gr

FROM Students St LEFT JOIN Groups G

ON G.Id\_Gr=St.Id\_Gr WHERE St.Id\_Gr is null