

SQL Advanced

- INNER JOIN
- LEFT JOIN
- RIGHT JOIN
- FULL OUTER JOIN
- CROSS JOIN
- UNION

Objective

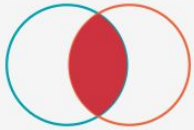
Be able to provide answer to meaningful questions by connecting the information between separate tables using JOIN clauses:

- Understand difference between different JOIN queries;
- Learn the JOIN query syntax;
- Be able to select which JOIN query is appropriate;
- Learn to combine information from different tables;
- Be able to combine JOIN, condition and aggregation functions in one query

SQL Advanced: Types of JOINS

INNER JOIN

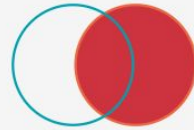
(or JOIN)



LEFT JOIN

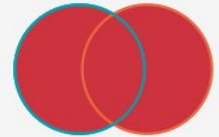


RIGHT JOIN



OUTER JOIN

(with UNION)



CROSS JOIN

*SQLite doesn't directly support the RIGHT JOIN and FULL OUTER JOIN

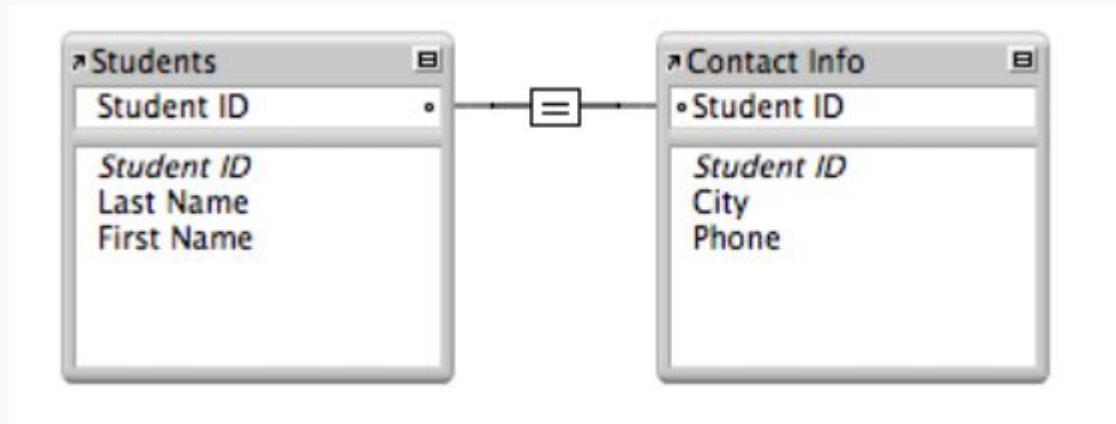
<https://joins.spathon.com/>

SQL Advanced: Types of Relationship

- **One-to-one relationship**
- **One-to-many relationship** (or many-to-one)
- **Many-to-many relationship**

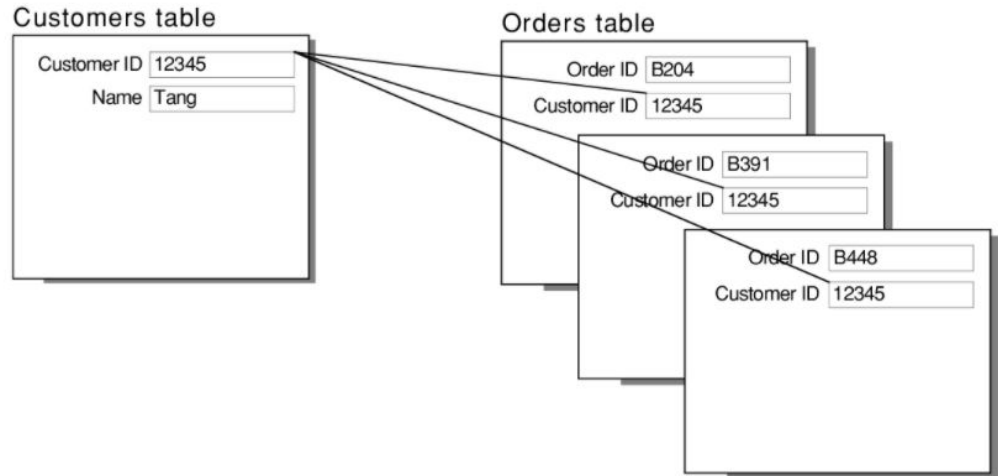
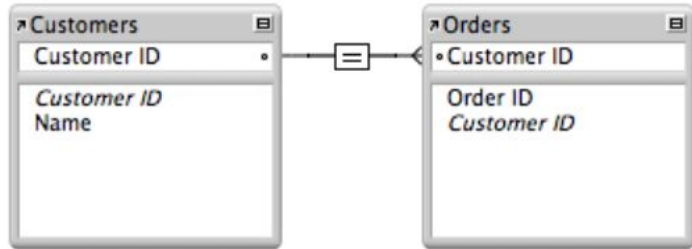
SQL Advanced: Types of Relationship: One-to-one

- In a **one-to-one relationship**, one record in a table is associated with one and only one record in another table.
- For example, in a school database, each student has only one student ID, and each student ID is assigned to only one person.



SQL Advanced: Types of Relationship: One-to-many

- In a **one-to-many relationship**, one record in a table can be associated with one or more records in another table.
- But one record from a second table is associated to a single record from the first table
- For example, each customer can have many sales orders.

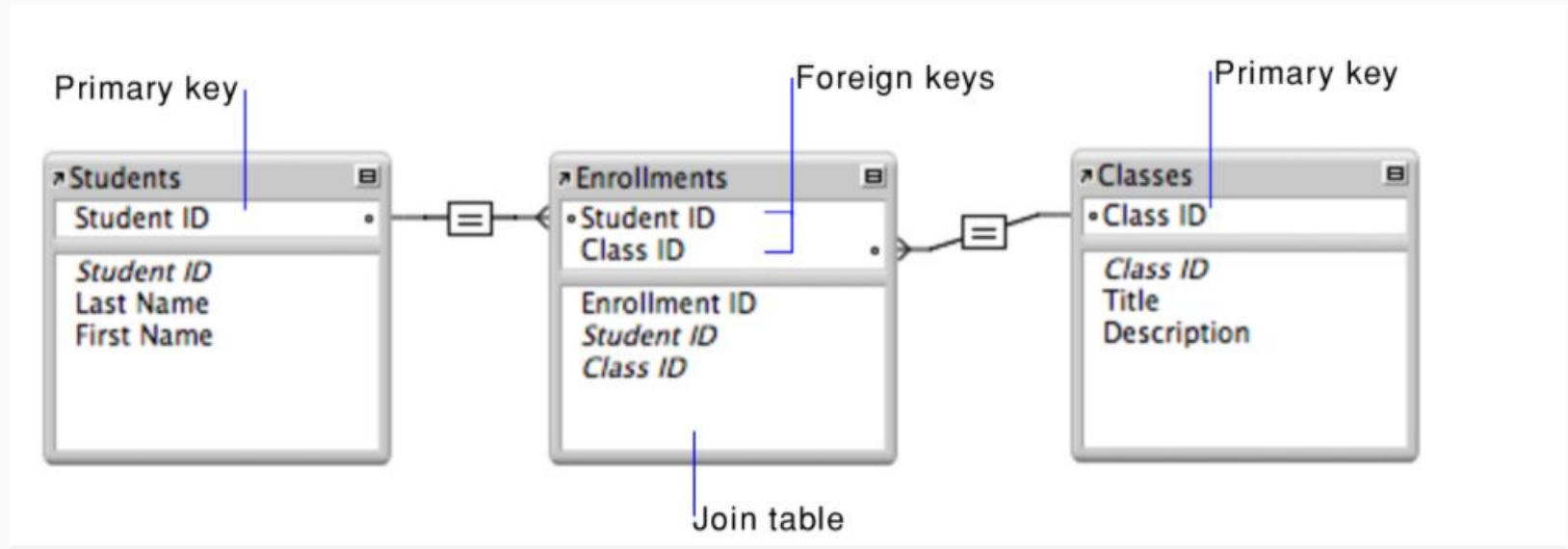


SQL Advanced: Types of Relationship: Many-to-many

- A **many-to-many relationship** occurs when multiple records in a table are associated with multiple records in another table.
- For example, a many-to-many relationship exists between customers and products: customers can purchase various products, and products can be purchased by many customers.
- In relational databases many-to-many relationship is usually represented by two one-to-many relationships by using a third table.
- Each record in a join table includes a match field that contains the value of the **primary keys** of the two tables it joins. (In the join table, these match fields are **foreign keys**.) These foreign key fields are populated with data as records in the join table are created from either table it joins.

SQL Advanced: Types of Relationship: Many-to-many

- Another typical example - students and classes: a student can register for many classes, and a class can include many students.



SQL Advanced: BROKERAGE DATASET - Types of Relationships

AccountManagers

Column

- AccountManagerID INTEGER
- AccountManager TEXT
- AccountManagerPhone TEXT

Customers

Column

- CustomerID INTEGER
- Customer INTEGER
- CustomerEmail TEXT
- CustomerSince TEXT
- AccountManagerID INTEGER
- CountryResidence TEXT
- CustomerClass TEXT
- AccountManager TEXT
- AccountManagerPhone TEXT
- PermitIDNeed TEXT
- CustomerAge INTEGER

PortfolioAmounts

Column

- AmountPosition REAL
- CustomerID INTEGER
- PortfolioPositionsID INTEGER

PortfolioPositions

Column

- PortfolioPositionsID INTEGER
- PortfolioPosition TEXT
- PositionType TEXT
- StockType TEXT

A

B

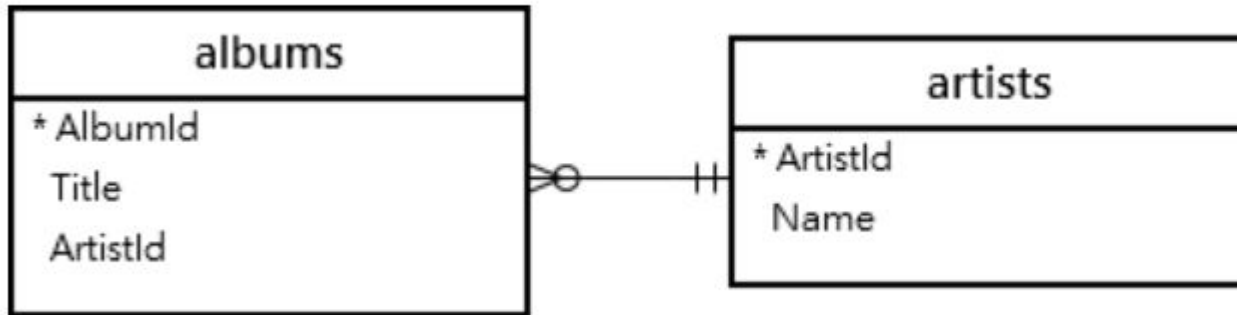
C

D



SQL Advanced: Demo dataset

- Artists and albums tables: an artist can have zero or many albums while an album belongs to one artist.



SQL Advanced: INNER JOIN (JOIN)



- INNER JOIN or JOIN returns records that match in **BOTH** right and left tables.
- INNER JOIN clause matches each row from the albums table with every row from the artists table based on the join condition (artists.ArtistId = albums.ArtistId) specified after the ON keyword.

```
SELECT
    Title,
    Name
FROM
    albums
INNER JOIN artists
    ON artists.ArtistId = albums.ArtistId;
```

SQL Advanced: INNER JOIN (JOIN) - aliases



- This query uses table **aliases** (l for the albums table and r for artists table) to shorten the query:

```
SELECT
  l.Title,
  r.Name
FROM
  albums l
INNER JOIN artists r ON
  r.ArtistId = l.ArtistId;
```

Table aliases

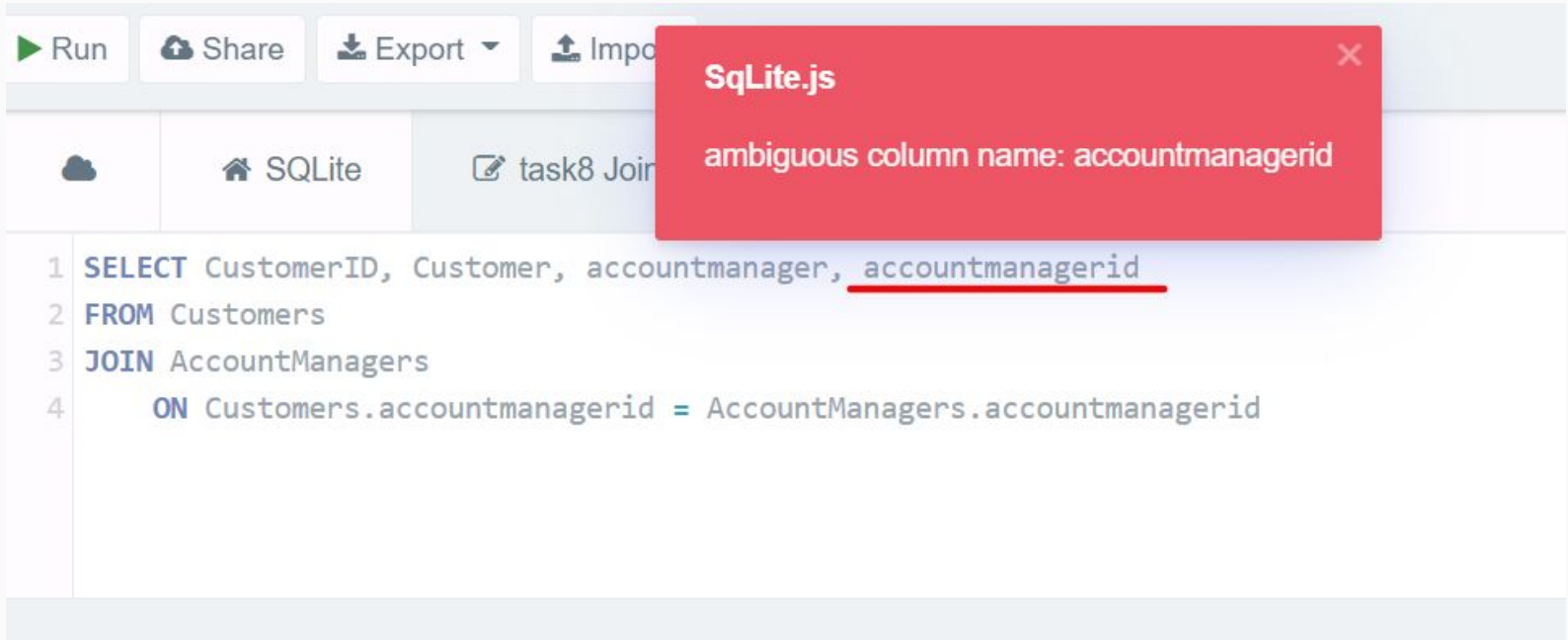
SQL Advanced: INNER JOIN (JOIN) - USING syntax



- In case the column names of joined tables are the same e.g., ArtistId, you can use the **USING** syntax
- The clause `USING(ArtistId)` is equivalent to the clause `ON artists.ArtistId = albums.ArtistId`

```
SELECT
    Title,
    Name
FROM
    albums
INNER JOIN artists USING(ArtistId);
```

SQL Advanced: JOIN pitfalls

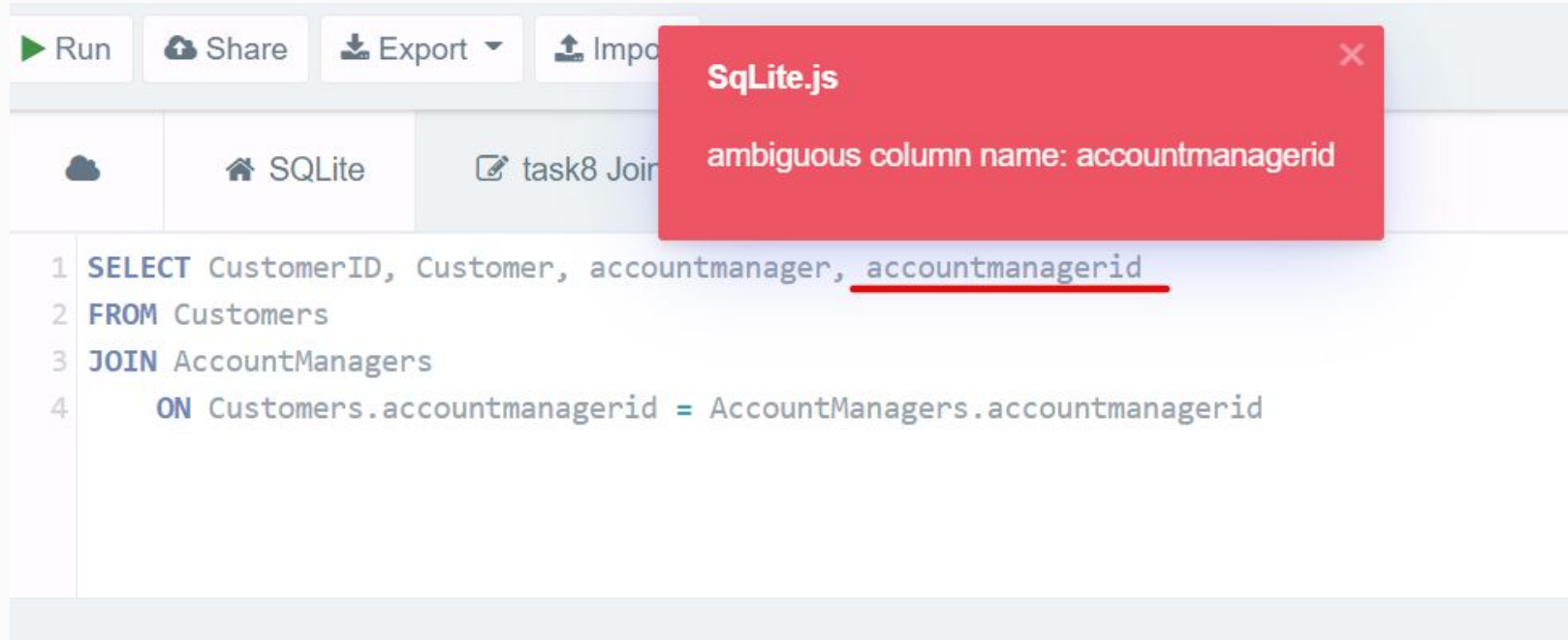


The screenshot shows a SQL editor interface with a red error message box overlaid on the query. The error message reads: "SQLite.js ambiguous column name: accountmanagerid". The query below is:

```
1 SELECT CustomerID, Customer, accountmanager, accountmanagerid
2 FROM Customers
3 JOIN AccountManagers
4 ON Customers.accountmanagerid = AccountManagers.accountmanagerid
```

The error occurs because the column `accountmanagerid` is not explicitly qualified with a table name in the `SELECT` clause, but it is used in the `ON` clause. This creates an ambiguity for the database engine.

SQL Advanced: JOIN pitfalls



```
1 SELECT CustomerID, Customer, accountmanager, accountmanagerid
2 FROM Customers
3 JOIN AccountManagers
4 ON Customers.accountmanagerid = AccountManagers.accountmanagerid
```

Because both Customers and Account Managers have same columnname query returns an error:

>> SOLUTION 1: use USING syntax

>> SOLUTION 2: use aliases

SQL Advanced: LEFT JOIN (LEFT OUTER JOIN)



- The **left join** returns all rows from the *left* table and the matching rows from the right table
- If a row from the left table doesn't have a matching row in the right table, SQLite includes columns of the rows in the left table and **NULL** for the columns of the right table..
- Similar to the INNER JOIN clause, you can use the **USING** syntax for the join condition

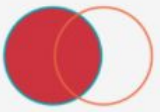
```
SELECT
    Name,
    Title
FROM
    artists
LEFT JOIN albums ON
    artists.ArtistId = albums.ArtistId
ORDER BY Name;
```


SQL Advanced: LEFT JOIN (LEFT OUTER JOIN)



- LEFT JOIN clause selects data starting from the left table (artists) and matching rows in the right table (albums) based on the join condition (artists.ArtistId = albums.ArtistId)
- **Where there is more than one match the left table record is duplicated!**

Artists:	Albums:
Name	Title
A Cor Do Som	[NULL] no match
AC/DC	For Those About To Rock We Salute You
AC/DC more than 1 match	Let There Be Rock
Aaron Copland & London Symphony Orchestra	A Copland Celebration, Vol. I
Aaron Goldberg	Worlds
Academy of St. Martin in the Fields & Sir Neville Marriner	The World of Classical Favourites
Academy of St. Martin in the Fields Chamber Ensemble &	Sir Neville Marriner: A Celebration
Academy of St. Martin in the Fields, John Birch, Sir Neville	Fauré: Requiem, Ravel: Pavane & Others
Academy of St. Martin in the Fields, Sir Neville Marriner &	Bach: Orchestral Suites Nos. 1 - 4
Academy of St. Martin in the Fields, Sir Neville Marriner &	[NULL] no match
Accept	Balls to the Wall
Accept more than 1 match	Restless and Wild
Adrian Leaper & Doreen de Feis	Górecki: Symphony No. 3



- If you want to find artists who don't have any albums, you can add a WHERE clause as shown in the following query

```
SELECT
    Name,
    Title
FROM
    artists
LEFT JOIN albums ON
    artists.ArtistId = albums.ArtistId
WHERE Title IS NULL
ORDER BY Name;
```

SQL Advanced: RIGHT JOIN (RIGHT OUTER JOIN)



- RIGHT JOIN is not supported in SQLite
- We can get same results just by switching the places of the right and left tables

>> What will be the output result in our demo _artist and _album case?



- Which artist has the biggest number of albums?
- How many albums does he have? Who is on the first and second place?
- How many artists don't have any albums?

SQL Advanced: JOIN

Rule of Thumb - Operations over JOINED tabled

1. Think which table should be on the LEFT side if using LEFT JOIN.
2. Will you lose some records?
3. Test the query step-by-step:
 - a. Check first if the join is implemented correctly
 - b. Add the WHERE clause
 - c. Start aggregating if needed

SQL Advanced: Multiple JOINS

Syntax:

```
SELECT ....
```

```
FROM ....
```

```
LEFT JOIN ...
```

```
    ON ..... = .....
```

```
LEFT JOIN ...
```

```
    ON ..... = .....
```

Question: JOIN other information to PortfolioPositions

Do we have all customers listed?

SQL Advanced: BROKERAGE DATASET

AccountManagers

Column

- AccountManagerID INTEGER
- AccountManager TEXT
- AccountManagerPhone TEXT

100%

Customers

Column

- CustomerID INTEGER
- Customer INTEGER
- CustomerEmail TEXT
- CustomerSince TEXT
- AccountManagerID INTEGER
- CountryResidence TEXT
- CustomerClass TEXT
- AccountManager TEXT
- AccountManagerPhone TEXT
- PermitIDNeed TEXT
- CustomerAge INTEGER

?%

PortfolioAmounts

Column

- AmountPosition REAL
- CustomerID INTEGER
- PortfolioPositionsID INTEGER

?%

PortfolioPositions

Column

- PortfolioPositionsID INTEGER
- PortfolioPosition TEXT
- PositionType TEXT
- StockType TEXT



- The result of the **FULL OUTER JOIN** is a combination of a LEFT JOIN and a RIGHT JOIN.
- The result set of the full outer join has NULL values for every column of the table that does not have a matching row in the other table.
- For the matching rows, the FULL OUTER JOIN produces a single row with values from columns of the rows in both tables.

>> By combining LEFT JOIN and **UNION** we can emulate the FULL OUTER JOIN in SQLite.



- Because SQLite does not support the **RIGHT JOIN** clause, we use the **LEFT JOIN** clause in the second **SELECT** statement instead and switch the positions of the tables.
- The **UNION ALL** clause retains the duplicate rows from the result sets of both queries.
- The **DISTINCT** clause removes rows that are duplicated



- If you use a LEFT JOIN, INNER JOIN, or CROSS JOIN without the ON or USING clause, SQLite produces the Cartesian product of the involved tables. The number of rows in the Cartesian product is the product of the number of rows in each involved tables.

```
SELECT *  
FROM A JOIN B;
```

```
SELECT *  
FROM A  
INNER JOIN B;
```

```
SELECT *  
FROM A  
CROSS JOIN B;
```

```
SELECT *  
FROM A, B;
```

Contacts:

www.linkedin.com/in/sergeyvicev/

vicev.sergey@gmail.com



THANK YOU