جامعة القاهرة الجديدة التكنولوجية

NEW CAIRO TECHNOLOGICAL UNIVERSITY

كلية تكنولوجيا الصناعة والطاقة

# PEARSON BTEC International Standards Verifier

ICT Program

Dr. Amany AbdElSamea Saeed

Lec. 5 MySQL-Constraints, User Management Privileges and Joins

4/18/2024 6:18 AM

## Outline

MySQL Constraints

Types of MySQL Constraints

User Management and Privileges

MySQL Joins

## MYSQL Constraints

## MySQL Constraints

- The constraint in MySQL is used to specify the rule that allows or restricts what values/data will be stored in the table.
- They provide a suitable method to ensure data accuracy and integrity inside the table. The term data integrity simply means that the data stored in the table is valid.
- It also helps to limit the type of data that will be inserted inside the table.
- If any interruption occurs between the constraint and data action, the action is failed.

## Types of MySQL Constraints

Constraints in MySQL is classified into two types:

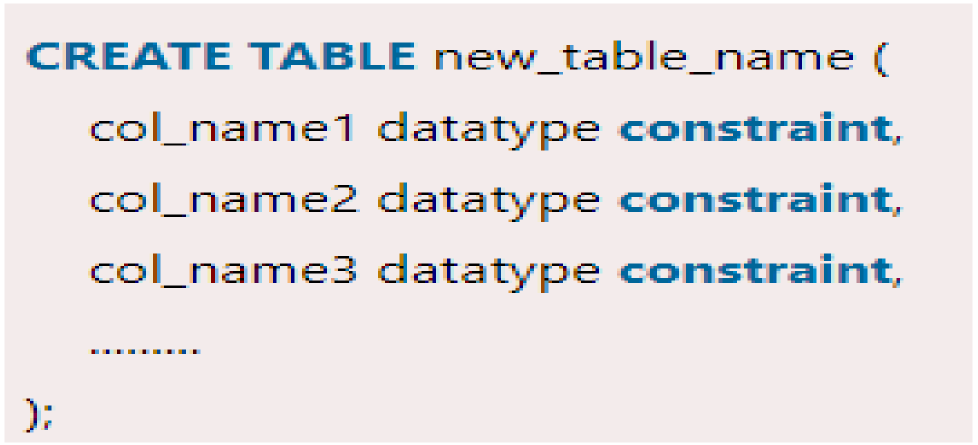
- **Column Level Constraints:** These constraints are applied only to the single column that limits the type of particular column data.

- **Table Level Constraints:** These constraints are applied to the entire table that limits the type of data for the whole table.

## How to create constraints in MySQL?

- We can define the constraints during a table created by using the *CREATE TABLE* statement. MySQL also uses the *ALTER TABLE* statement to specify the constraints in the case of the existing table schema.

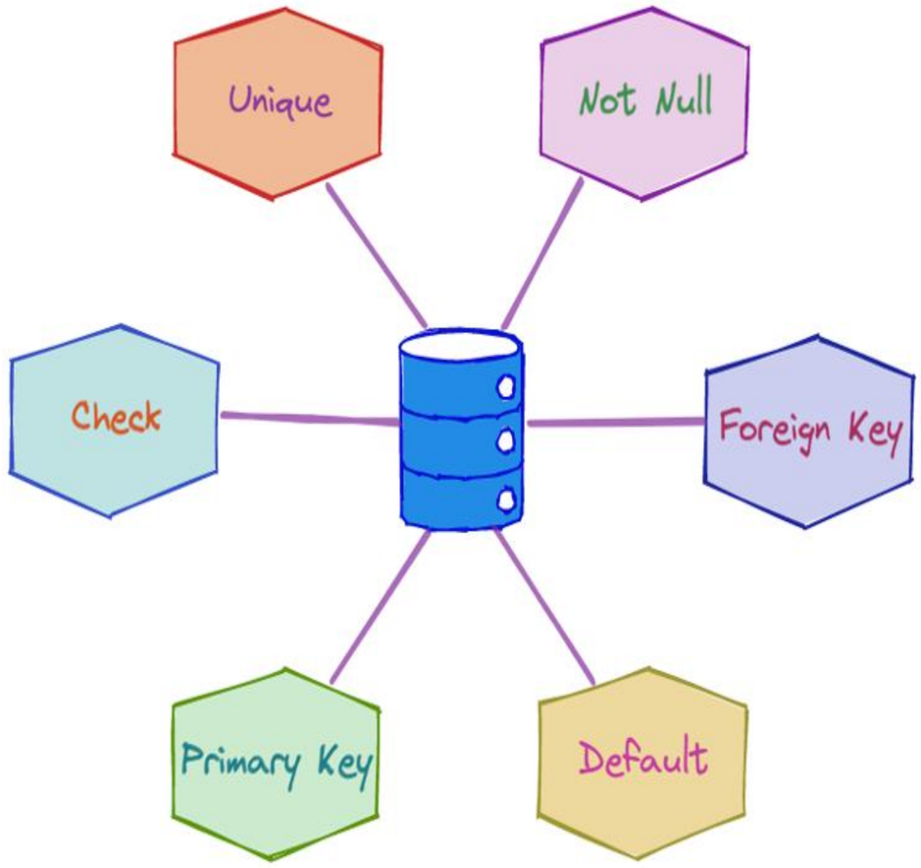
Syntax:



```
CREATE TABLE new_table_name (
    col_name1 datatype constraint,
    col_name2 datatype constraint,
    col_name3 datatype constraint,
    .........
);
```

# Constraints used in MySQL

- NOT NULL Constraint – Ensures that a column cannot have NULL value.
- DEFAULT Constraint – Provides a default value for a column when none is specified.
- UNIQUE Constraint – Ensures that all values in a column are different.
- PRIMARY Key – Uniquely identifies each row/record in a database table.
- FOREIGN Key – The foreign key is used to link one or more than one table together.
- CHECK Constraint – The CHECK constraint ensures that all the values in a column satisfies certain conditions.
- INDEX – Used to create and retrieve data from the database very quickly.



## NOT NULL Constraint

- By default, a column can hold NULL values.
- If you do not want a column to have a NULL value, then you need to define such a constraint on this column specifying that NULL is now not allowed for that column.
- Example:

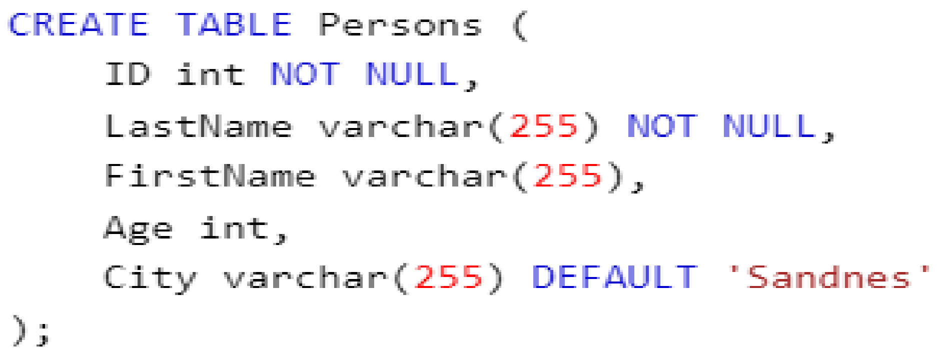
```
CREATE TABLE CUSTOMERS(
    ID    INT                    NOT NULL,
    NAME  VARCHAR (20)           NOT NULL,
    AGE   INT                    NOT NULL,
    ADDRESS   CHAR (25) ,
    SALARY    DECIMAL (18, 2),
    PRIMARY KEY (ID)
);
```

- If CUSTOMERS table has already been created, then to add a NOT NULL constraint to the SALARY column, you would write the following query

```
ALTER TABLE CUSTOMERS
   MODIFY SALARY  DECIMAL (18, 2) NOT NULL;
```

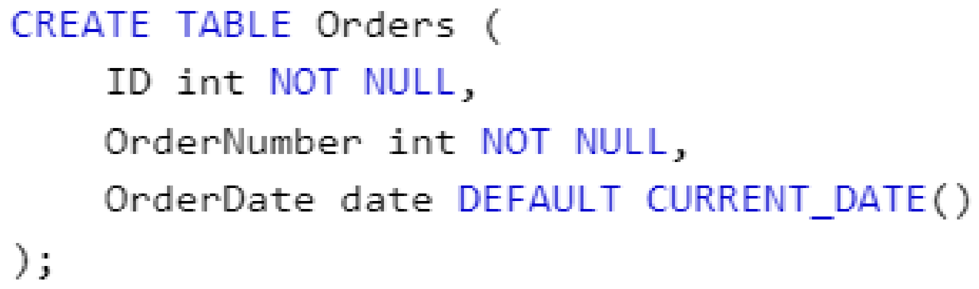
# DEFAULT Constraint

- The DEFAULT constraint provides a default value to a column when the INSERT INTO statement does not provide a specific value.
- Example:



```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    City varchar(255) DEFAULT 'Sandnes'
);
```

- The DEFAULT constraint can also be used to insert system values, by using functions like CURRENT_DATE()

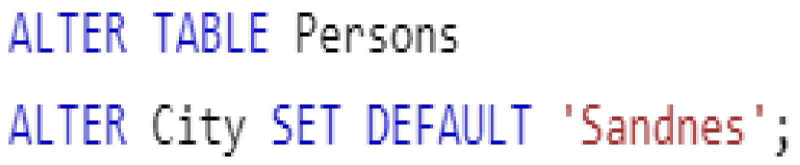


```
CREATE TABLE Orders (
    ID int NOT NULL,
    OrderNumber int NOT NULL,
    OrderDate date DEFAULT CURRENT_DATE()
);
```

جامعة القاهرة الجديدة التكنولوجية
NEW CAIRO TECHNOLOGICAL UNIVERSITY
كلية تكنولوجيا الصناعة والطاقة

# DEFAULT Constraint cont.,

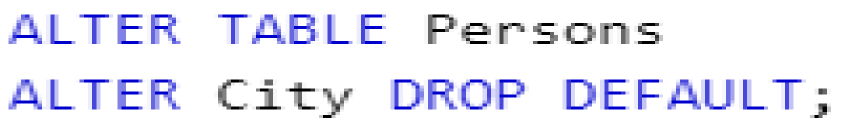
- To create a DEFAULT constraint on the "City" column when the table is already created, use the following SQL:



```
ALTER TABLE Persons
ALTER City SET DEFAULT 'Sandnes';
```

- DROP a DEFAULT Constraint

To drop a DEFAULT constraint, use the following SQL:



```
ALTER TABLE Persons
ALTER City DROP DEFAULT;
```

## Unique Constraint

- The UNIQUE Constraint prevents two records from having identical values in a column
- Here, the AGE column is set to UNIQUE, so that you cannot have two records with the same age

```
CREATE TABLE CUSTOMERS(
   ID   INT              NOT NULL,
   NAME VARCHAR (20)     NOT NULL,
   AGE  INT              NOT NULL UNIQUE,
   ADDRESS  CHAR (25) ,
   SALARY   DECIMAL (18, 2),
   PRIMARY KEY (ID)
);
```

- If the CUSTOMERS table has already been created, then to add a UNIQUE constraint to the AGE column

```
ALTER TABLE CUSTOMERS
   MODIFY AGE INT NOT NULL UNIQUE;
```

## Unique Constraint cont.,

- Naming the constraint in multiple columns

```
ALTER TABLE CUSTOMERS
   ADD CONSTRAINT myUniqueConstraint UNIQUE(AGE, SALARY);
```

- DROP a UNIQUE Constraint

```
ALTER TABLE CUSTOMERS
    DROP INDEX myUniqueConstraint;
```

## Primary Key Constraint

- A primary key is a field in a table which uniquely identifies each row/record in a database table.
- Primary keys must contain unique values.
- A primary key column cannot have NULL values.
- A table can have only one primary key, which may consist of single or multiple fields. When multiple fields are used as a primary key, they are called a composite key.
- If a table has a primary key defined on any field(s), then you cannot have two records having the same value of that field(s).

جامعة القاهرة الجديدة التكنولوجية NCT NEW CAIRO TECHNOLOGICAL UNIVERSITY

كلية تكنولوجيا الصناعة والطاقة

## Primary Key Constraint cont.,

- Create Primary Key

```
CREATE TABLE CUSTOMERS(
   ID   INT                NOT NULL,
   NAME VARCHAR (20)       NOT NULL,
   AGE  INT                NOT NULL,
   ADDRESS  CHAR (25) ,
   SALARY   DECIMAL (18, 2),
   PRIMARY KEY (ID)
);
```

- To create a PRIMARY KEY constraint on the "ID" column when the CUSTOMERS table already exists, use the following SQL syntax –

```
ALTER TABLE CUSTOMER ADD PRIMARY KEY (ID);
```

## Primary Key Constraint cont.,

- For defining a PRIMARY KEY constraint on multiple columns

```
CREATE TABLE CUSTOMERS(
   ID   INT                   NOT NULL,
   NAME VARCHAR (20)          NOT NULL,
   AGE  INT                   NOT NULL,
   ADDRESS  CHAR (25) ,
   SALARY   DECIMAL (18, 2),
   PRIMARY KEY (ID, NAME)
);
```

- To create a PRIMARY KEY constraint on the "ID" and "NAMES" columns when CUSTOMERS table already exists,

```
ALTER TABLE CUSTOMERS
   ADD CONSTRAINT PK_CUSTID PRIMARY KEY (ID, NAME);
```

- Delete Primary Key

```
ALTER TABLE CUSTOMERS DROP PRIMARY KEY ;
```

جامعة القاهرة الجديدة التكنولوجية NEW CAIRO TECHNOLOGICAL UNIVERSITY
كلية تكنولوجيا الصناعة والطاقة

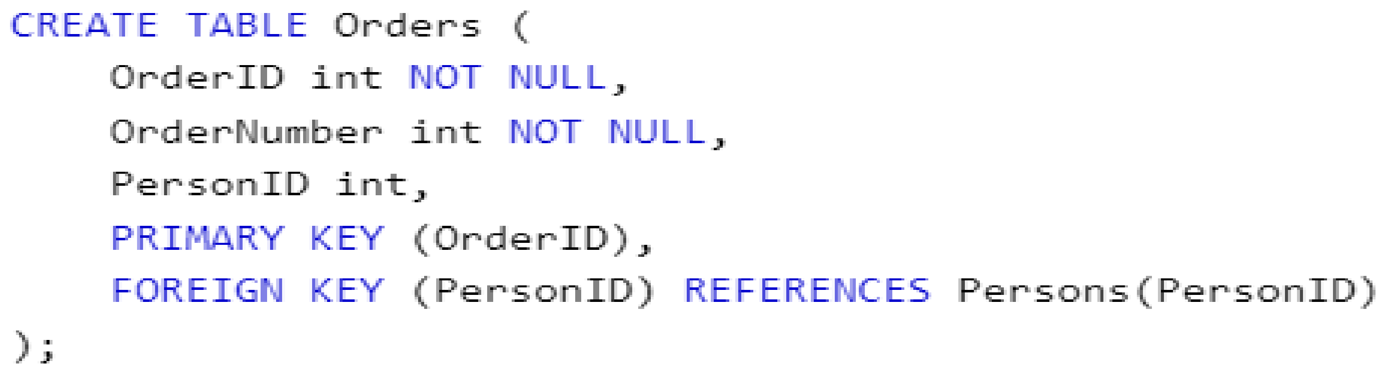
## Foreign Key Constraints

- A foreign key is a key used to link two tables together. This is sometimes also called as a referencing key.
- A Foreign Key is a column or a combination of columns whose values match a Primary Key in a different table.

**FOREIGN KEY on CREATE TABLE**

Notice that the "PersonID" column in the "Orders" table points to the "PersonID" column in the "Persons" table

- The following SQL creates a FOREIGN KEY on the "PersonID" column when the "Orders" table is created:



```
CREATE TABLE Orders (
    OrderID int NOT NULL,
    OrderNumber int NOT NULL,
    PersonID int,
    PRIMARY KEY (OrderID),
    FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)
);
```

# Foreign Key Constraint cont.,

- To allow naming of a FOREIGN KEY constraint, and for defining a FOREIGN KEY constraint on multiple columns

```
CREATE TABLE Orders (
    OrderID int NOT NULL,
    OrderNumber int NOT NULL,
    PersonID int,
    PRIMARY KEY (OrderID),
    CONSTRAINT FK_PersonOrder FOREIGN KEY (PersonID)
    REFERENCES Persons(PersonID)
);
```

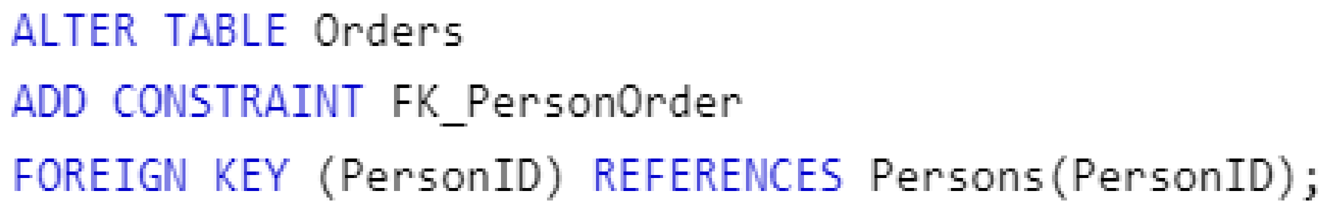
- To create a FOREIGN KEY constraint on the "PersonID" column when the "Orders" table is already created

```
ALTER TABLE Orders
ADD FOREIGN KEY (PersonID) REFERENCES Persons(PersonID);
```

كلية تكنولوجيا الصناعة والطاقة

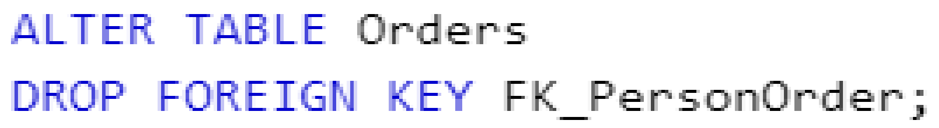
## Foreign Key Constraint cont.,

- To allow naming of a FOREIGN KEY constraint, and for defining a FOREIGN KEY constraint on multiple columns



```
ALTER TABLE Orders
ADD CONSTRAINT FK_PersonOrder
FOREIGN KEY (PersonID) REFERENCES Persons(PersonID);
```

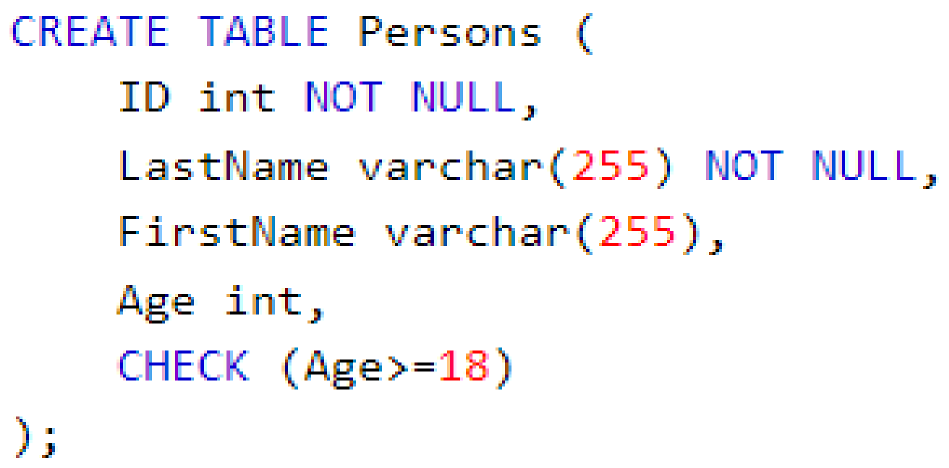
- DROP a FOREIGN KEY Constraint



```
ALTER TABLE Orders
DROP FOREIGN KEY FK_PersonOrder;
```

## Check Constraint

- The CHECK Constraint enables a condition to check the value being entered into a record. If the condition evaluates to false, the record violates the constraint and isn't entered the table.
- The following SQL creates a CHECK constraint on the "Age" column when the "Persons" table is created. The CHECK constraint ensures that the age of a person must be 18, or older:



```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    CHECK (Age>=18)
);
```

## Check Constraint cont.,

- To allow naming of a CHECK constraint, and for defining a CHECK constraint on multiple columns

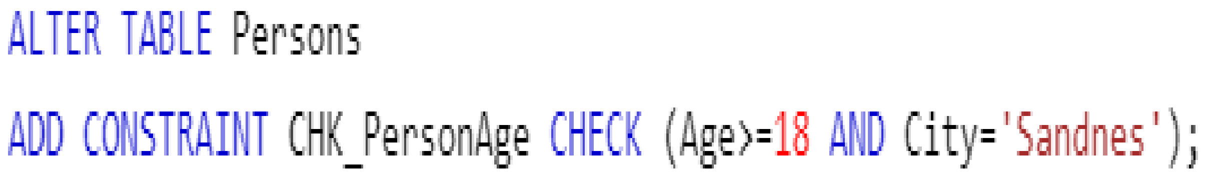
```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    City varchar(255),
    CONSTRAINT CHK_Person CHECK (Age>=18 AND City='Sandnes')
);
```

- To create a CHECK constraint on the "Age" column when the table is already created,

```
ALTER TABLE Persons
ADD CHECK (Age>=18);
```

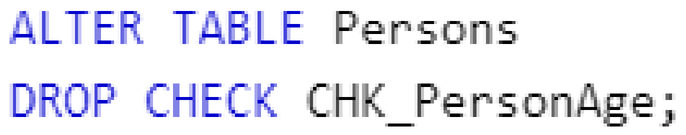
## Check Constraint cont.,

- To allow naming of a CHECK constraint, and for defining a CHECK constraint on multiple columns



```
ALTER TABLE Persons
ADD CONSTRAINT CHK_PersonAge CHECK (Age>=18 AND City='Sandnes');
```

- DROP a CHECK Constraint



```
ALTER TABLE Persons
DROP CHECK CHK_PersonAge;
```

## Index

- The INDEX is used to create and retrieve data from the database very quickly. An Index can be created by using a single or group of columns in a table. When the index is created, it is assigned a ROWID for each row before it sorts out the data.
- Syntax:

```
CREATE INDEX index_name
   ON table_name ( column1, column2.....);
```

- To create an INDEX on the AGE column, to optimize the search on customers for a specific age

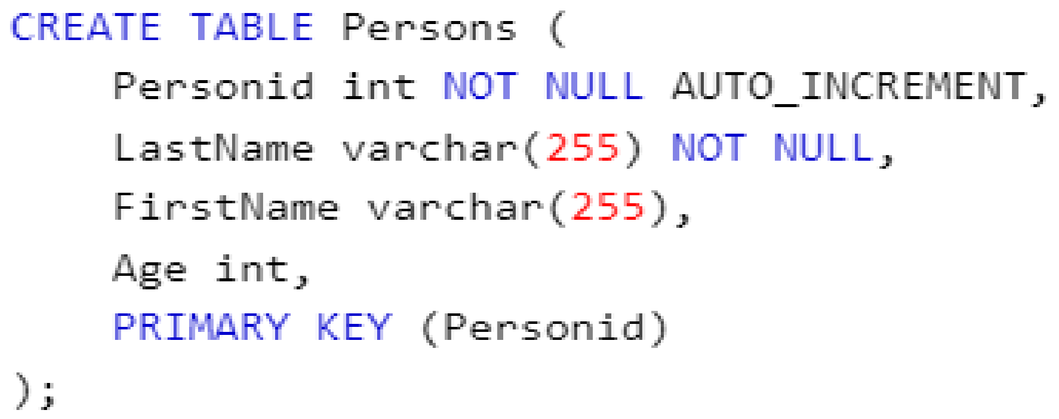
```
CREATE INDEX idx_age
   ON CUSTOMERS ( AGE );
```

- DROP an INDEX Constraint

```
ALTER TABLE CUSTOMERS
   DROP INDEX idx_age;
```

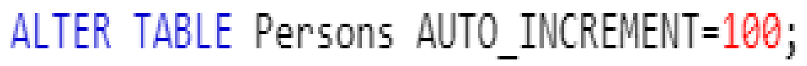
## Auto Increment

- MySQL uses the AUTO_INCREMENT keyword to perform an auto-increment feature.
- By default, the starting value for AUTO_INCREMENT is 1, and it will increment by 1 for each new record.



```
CREATE TABLE Persons (
    Personid int NOT NULL AUTO_INCREMENT,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    PRIMARY KEY (Personid)
);
```

- To let the AUTO_INCREMENT sequence start with another value



```
ALTER TABLE Persons AUTO_INCREMENT=100;
```

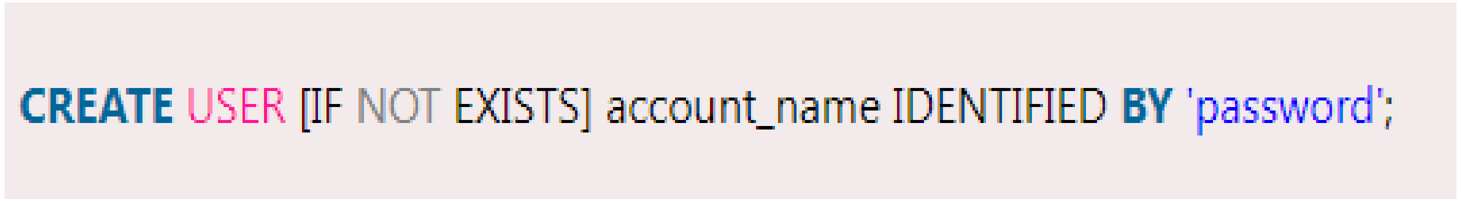
## User Management and Privileges

جامعة القاهرة الجديدة التكنولوجية NEW CAIRO TECHNOLOGICAL UNIVERSITY

كلية تكنولوجيا الصناعة والطاقة

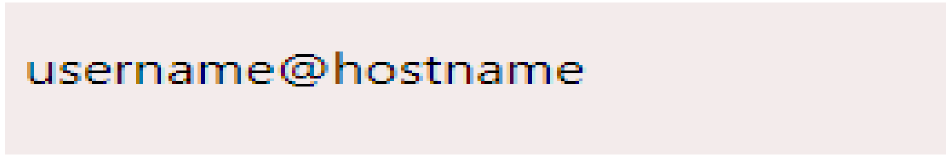
## User Management

- The MySQL user is a record in the **USER** table of the MySQL server that contains the login information, account privileges, and the host information for MySQL account.
- It is essential to create a user in MySQL for accessing and managing the databases.
- MySQL Create User



```
CREATE USER [IF NOT EXISTS] account_name IDENTIFIED BY 'password';
```

- In the above syntax, the **account_name** has two parts one is the **username**, and another is the **hostname**, which is separated by **@** symbol. Here, the username is the name of the user, and the hostname is the name of the host from which the user can connect with the database server.

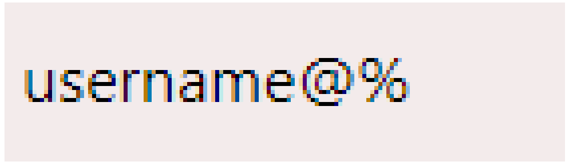


```
username@hostname
```

جامعة القاهرة الجديدة التكنولوجية NEW CAIRO TECHNOLOGICAL UNIVERSITY
كلية تكنولوجيا الصناعة والطاقة

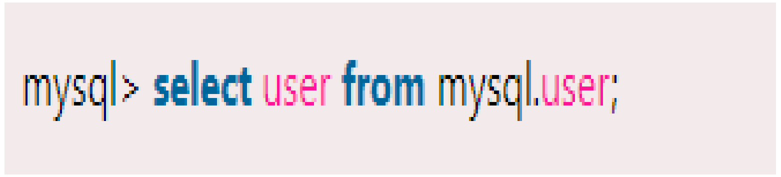
# User Management cont.,

- The hostname is optional. If you have not given the hostname, the user can connect from any host on the server. The user account name without hostname can be written as:

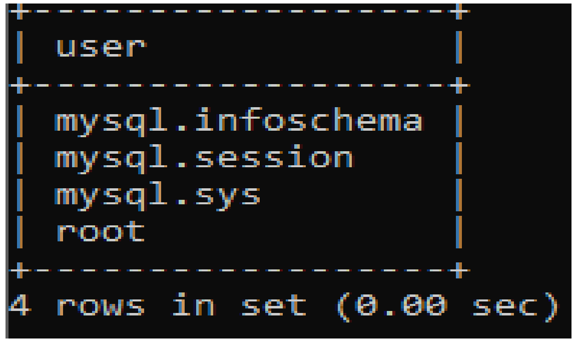


```
username@%
```

- Execute the following command to show all users in the current MySQL server.



```
mysql> select user from mysql.user;
```



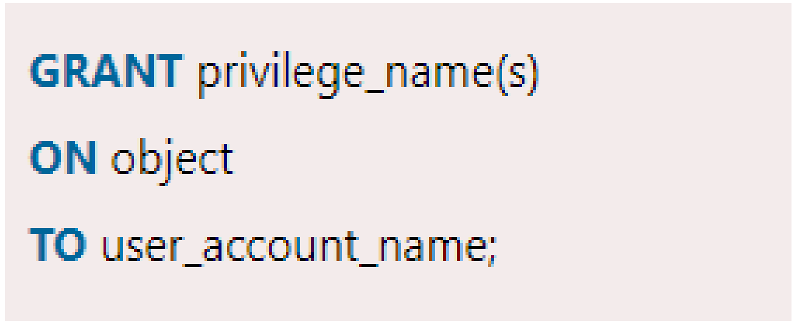
```
+-------------------+
| user              |
+-------------------+
| mysql.infoschema  |
| mysql.session     |
| mysql.sys         |
| root              |
+-------------------+
4 rows in set (0.00 sec)
```

## Grant Privileges to the MySQL New User

- MySQL server provides multiple types of privileges to a new user account. Some of the most commonly used privileges are given below:
  - **ALL PRIVILEGES:** It permits all privileges to a new user account.
  - **CREATE:** It enables the user account to create databases and tables.
  - **DROP:** It enables the user account to drop databases and tables.
  - **DELETE:** It enables the user account to delete rows from a specific table.
  - **INSERT:** It enables the user account to insert rows into a specific table.
  - **SELECT:** It enables the user account to read a database.
  - **UPDATE:** It enables the user account to update table rows.

## GRANT Statement

- The grant statement enables system administrators to ***assign privileges and roles*** to the MySQL user accounts so that they can use the assigned permission on the database whenever required.
- Syntax
- The following are the basic syntax of using the GRANT statement:



```
GRANT privilege_name(s)
ON object
TO user_account_name;
```

## GRANT Privileges

- If you want to give all privileges to a newly created user, execute the following command

```
mysql> GRANT ALL PRIVILEGES ON * . * TO peter@localhost;
```

- If you want to give specific privileges to a newly created user, execute the following command

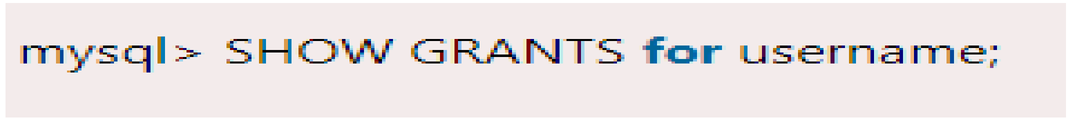
```
mysql> GRANT CREATE, SELECT, INSERT ON * . * TO peter@localhost;
```

- Sometimes, you want to **flush** all the privileges of a user account for changes occurs immediately, type the following command. **To tell the server to reload the grant tables**

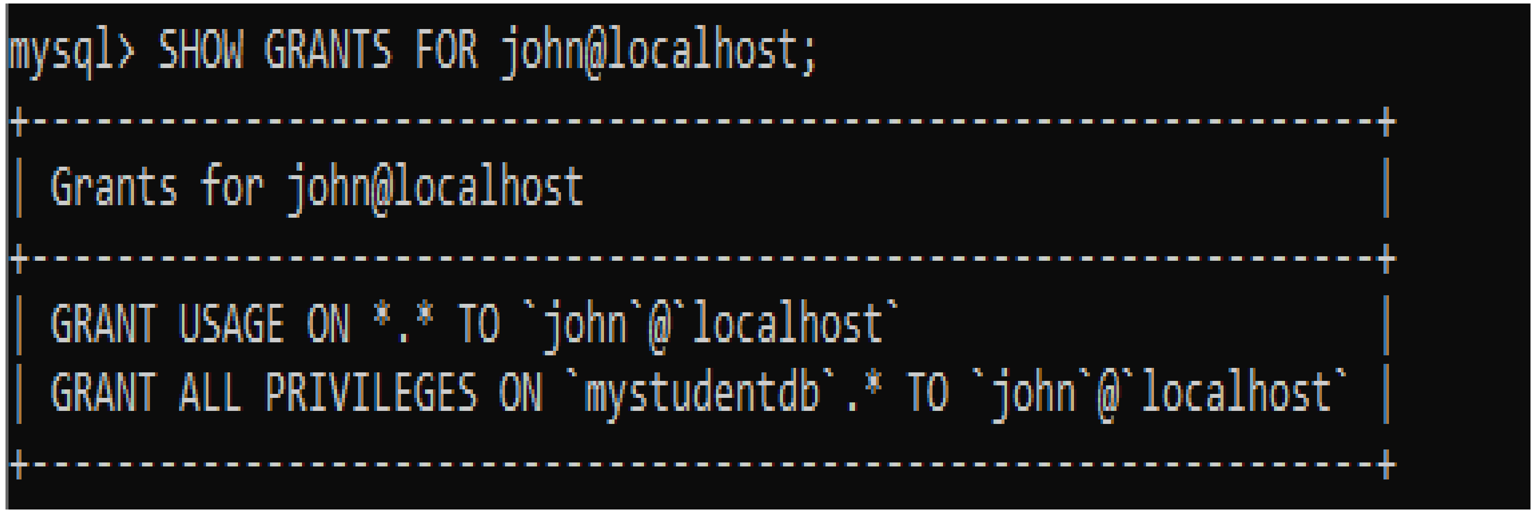
```
FLUSH PRIVILEGES;
```

## Show GRANTs

If you want to see the existing privileges for the user, execute the following command.



```
mysql> SHOW GRANTS for username;
```



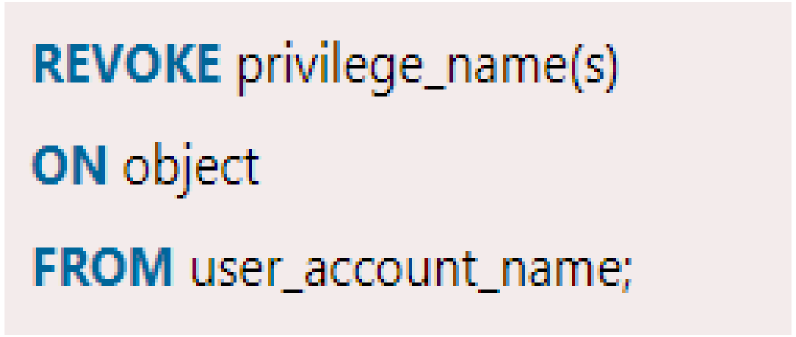
```
mysql> SHOW GRANTS FOR john@localhost;
+-------------------------------------------------------------+
| Grants for john@localhost                                   |
+-------------------------------------------------------------+
| GRANT USAGE ON *.* TO `john`@`localhost`                    |
| GRANT ALL PRIVILEGES ON `mystudentdb`.* TO `john`@`localhost` |
+-------------------------------------------------------------+
```

# REVOKE Statement

- The revoke statement enables system administrators to ***revoke privileges and roles*** to the MySQL user accounts so that they cannot use the assigned permission on the database in the past.

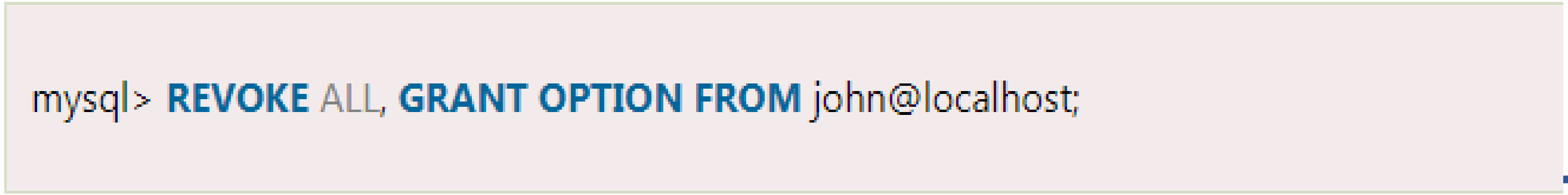
Syntax

- The following are the basic syntax of using the REVOKE statement:



```
REVOKE privilege_name(s)
ON object
FROM user_account_name;
```

If we want to revoke all privileges assign to the user, execute the following statement:



```
mysql> REVOKE ALL, GRANT OPTION FROM john@localhost;
```

## Drop User

- The MySQL Drop User statement allows us to **remove** one or more user accounts and their **privileges** from the database server. If the account does not exist in the database server, it gives an error.
- If you want to use the Drop User statement, it is required to have a **global** privilege of Create User statement or the **DELETE** privilege for the MySQL system schema.

Syntax:

- The following syntax is used to delete the user accounts from the database server completely.

```
DROP USER 'account_name';
```

- The **account_name** can be identified with the following syntax:

```
username@hostname
```

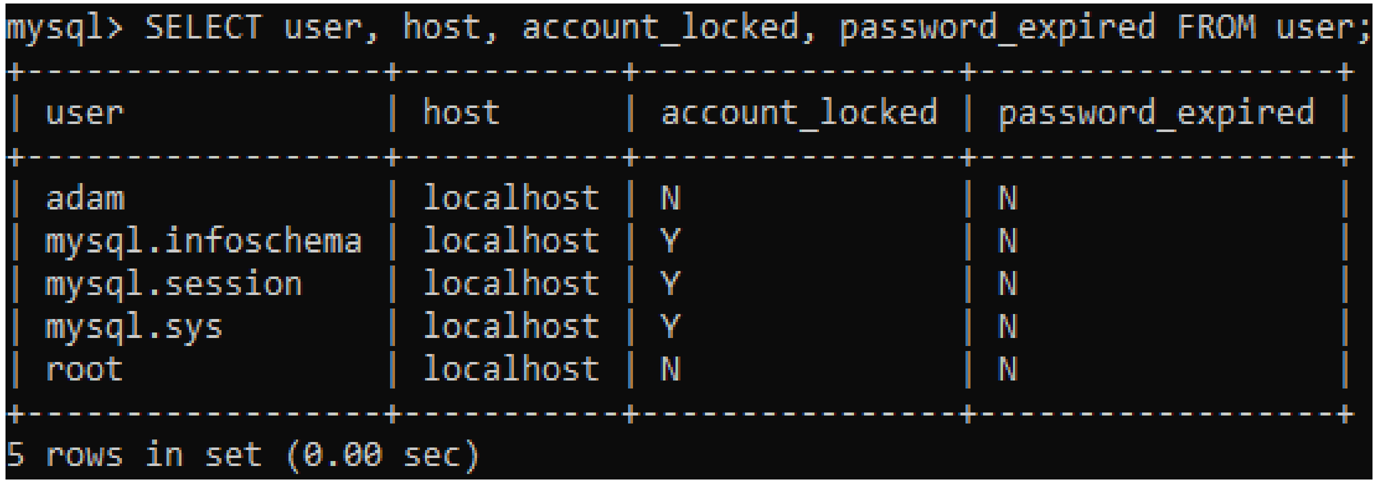
## Show Users

- We can use the following query to see the list of all user in the database server

```
mysql> Select user from mysql.user;
```

- To get the selected information like as hostname, password expiration status, and account locking

```
mysql> SELECT user, host, account_locked, password_expired FROM user;
```



```
mysql> SELECT user, host, account_locked, password_expired FROM user;
+------------------+-----------+----------------+------------------+
| user             | host      | account_locked | password_expired |
+------------------+-----------+----------------+------------------+
| adam             | localhost | N              | N                |
| mysql.infoschema | localhost | Y              | N                |
| mysql.session    | localhost | Y              | N                |
| mysql.sys        | localhost | Y              | N                |
| root             | localhost | N              | N                |
+------------------+-----------+----------------+------------------+
5 rows in set (0.00 sec)
```

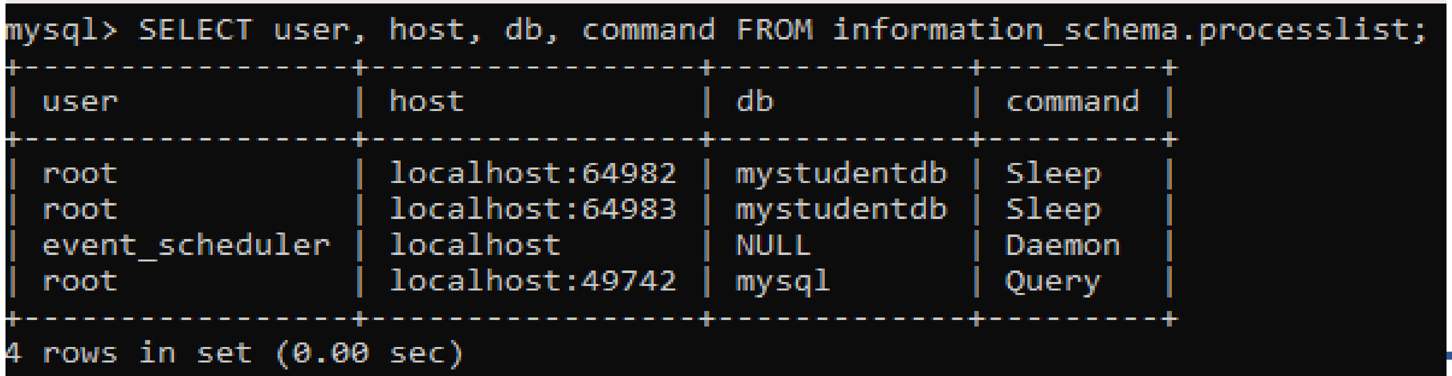
# Show Current Users

- We can get information of the current user by using the **user() or current_user()** function

```
mysql> Select user();
        or,
mysql> Select current_user();
```

- We can see the currently logged user in the database server by using the following query in the MySQL server:

```
mysql> SELECT user, host, db, command FROM information_schema.processlist;
```



```
mysql> SELECT user, host, db, command FROM information_schema.processlist;
+-----------------+-----------------+-------------+---------+
| user            | host            | db          | command |
+-----------------+-----------------+-------------+---------+
| root            | localhost:64982 | mystudentdb | Sleep   |
| root            | localhost:64983 | mystudentdb | Sleep   |
| event_scheduler | localhost       | NULL        | Daemon  |
| root            | localhost:49742 | mysql       | Query   |
+-----------------+-----------------+-------------+---------+
4 rows in set (0.00 sec)
```

جامعة القاهرة الجديدة التكنولوجية NEW CAIRO TECHNOLOGICAL UNIVERSITY
كلية تكنولوجيا الصناعة والطاقة

# Change MySQL User Password

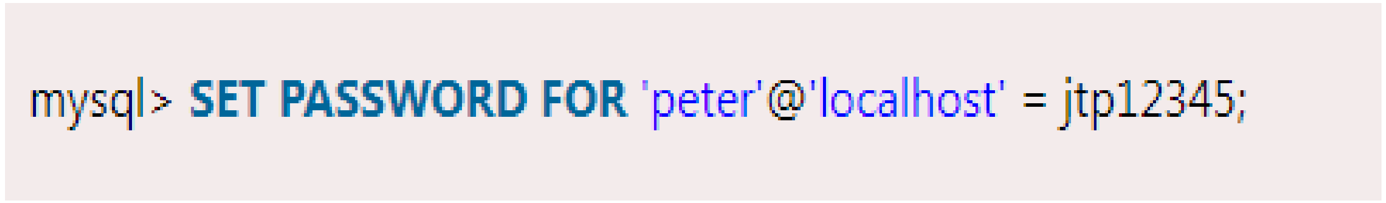
- MySQL allows us to change the user account password in three different ways, which are given below:
- UPDATE Statement
- SET PASSWORD Statement
- ALTER USER Statement

1- **UPDATE Statement** for MySQL version **5.7.6** or higher,

```
mysql> USE mysql;
mysql> mysql> UPDATE user SET authentication_string = PASSWORD('jtp12345') WHERE user = 'peter' AND host = 'localhost';
mysql> FLUSH PRIVILEGES;
```

## Change MySQL User Password cont.,

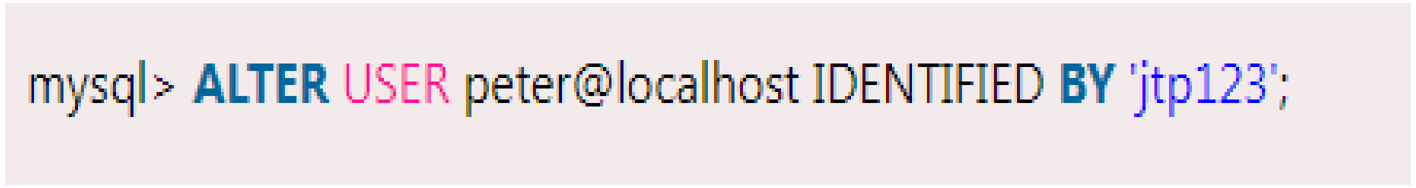
- **SET PASSWORD statement** for MySQL version **5.7.6** or higher



```
mysql> SET PASSWORD FOR 'peter'@'localhost' = jtp12345;
```

- **ALTER USER statement**

MySQL uses ALTER USER statement with the IDENTIFIED BY clause for changing the password of a user account.

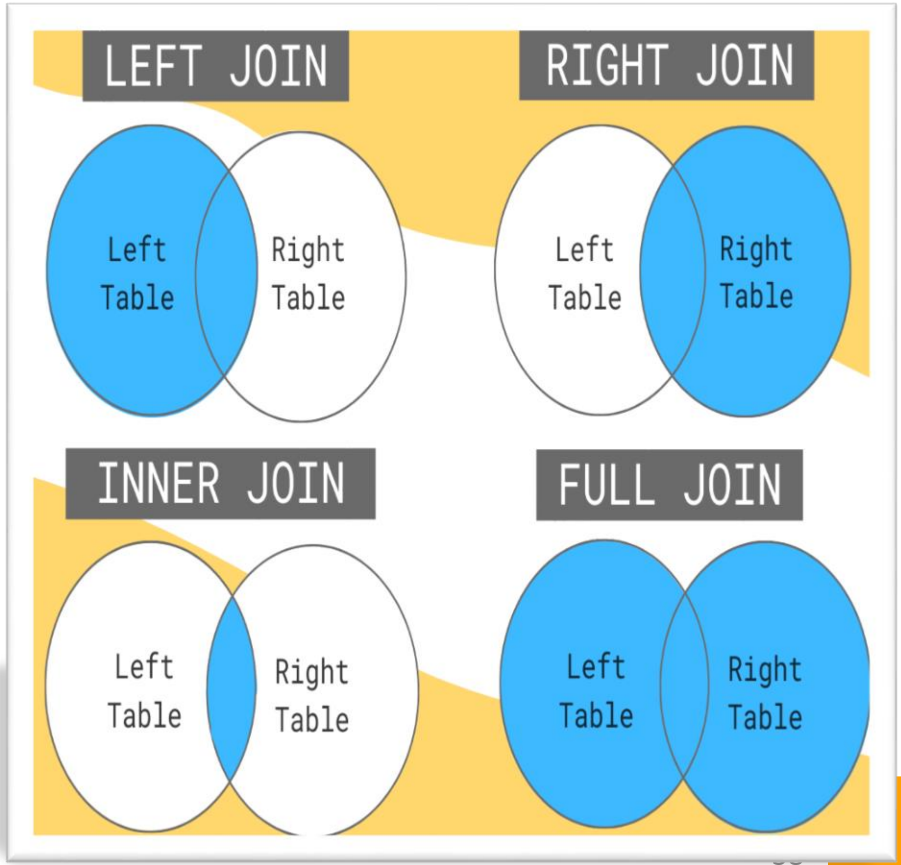


```
mysql> ALTER USER peter@localhost IDENTIFIED BY 'jtp123';
```

## MYSQL Joins

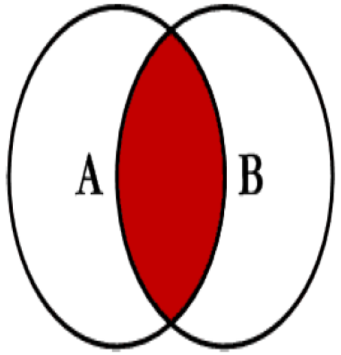
# Join

- **JOIN** is used to combine rows from two or more tables, based on a related column between them
- Different types of joins include:
  - Inner join
  - Left join
  - Right join
  - Cross join
  - Self join



# Inner Join

- The **inner join** command displays all rows from both the tables that have a common attribute.
- Inner join produces only the set of records that match in both Table A and Table B
- Most commonly used, best understood join



## Syntax

The basic syntax of the **INNER JOIN** is as follows.

```
SELECT table1.column1, table2.column2...
FROM table1
INNER JOIN table2
ON table1.common_field = table2.common_field;
```

# Inner Join Example

Consider two tables: Course and instructor

```
SELECT * FROM course INNER JOIN instructor ON course. Course_id
= instructor. Course_id;
```

| Course_id | Course |
|---|---|
| 111 | Science |
| 222 | Finance |
| 333 | History |
| 444 | Art |
| 555 | Music |
| 777 | Physics |

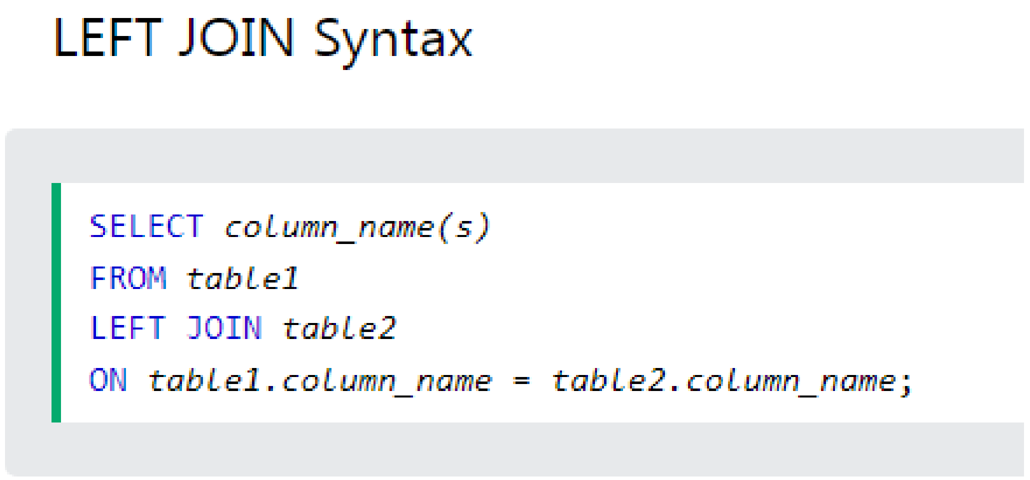
| Instructor_id | Name | Place | Course_id |
|---|---|---|---|
| 1 | Lisa | California | 111 |
| 2 | Roy | Bangalore | 222 |
| 3 | Mathew | Mumbai | 333 |
| 4 | Lorna | Boston | 444 |
| 5 | Marcelo | Texas | 555 |
| 6 | Pinto | Delhi | 666 |

| Course_id | Course | Instructor_id | Name | Place | Course_id |
|---|---|---|---|---|---|
| 111 | Science | 1 | Lisa | California | 111 |
| 222 | Finance | 2 | Roy | Bangalore | 222 |
| 333 | History | 3 | Mathew | Mumbai | 333 |
| 444 | Art | 4 | Lorna | Boston | 444 |
| 555 | Music | 5 | Marcelo | Texas | 555 |

جامعة القاهرة الجديدة التكنولوجية
NEW CAIRO TECHNOLOGICAL UNIVERSITY
كلية تكنولوجيا الصناعة والطاقة

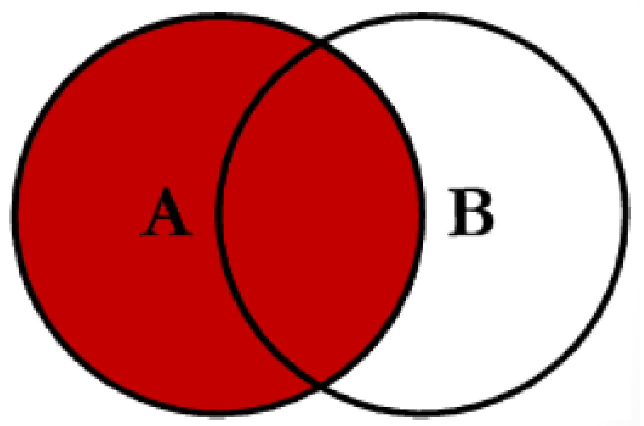
# Left Join

- The **left join** command displays the rows that have a matching attribute in both the tables along with the remaining rows of the table on the left side of the join.
- For the rows on the left table for which there is no matching value for the attribute on right table, the result table displays NULL.



LEFT JOIN Syntax

```
SELECT column_name(s)
FROM table1
LEFT JOIN table2
ON table1.column_name = table2.column_name;
```



# Left Join Example

```
SELECT * FROM course LEFT JOIN instructor ON course. Course_id =
instructor. Course_id;
```

| Course_id | Course |
|---|---|
| 111 | Science |
| 222 | Finance |
| 333 | History |
| 444 | Art |
| 555 | Music |
| 777 | Physics |

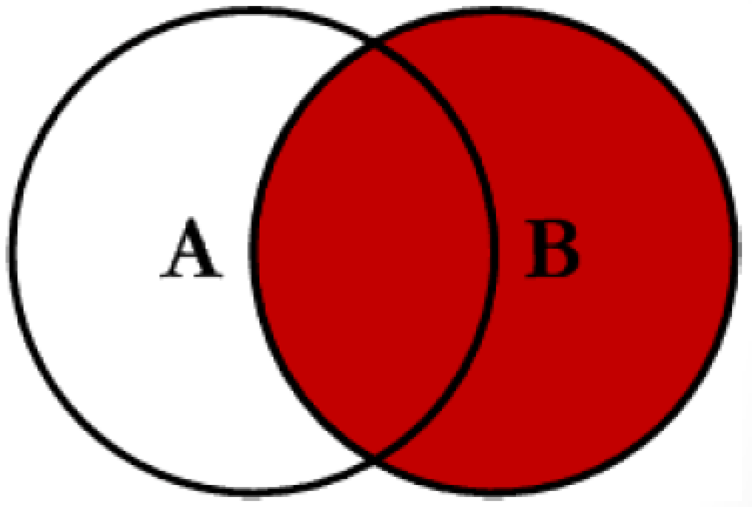
| Instructor_id | Name | Place | Course_id |
|---|---|---|---|
| 1 | Lisa | California | 111 |
| 2 | Roy | Bangalore | 222 |
| 3 | Mathew | Mumbai | 333 |
| 4 | Lorna | Boston | 444 |
| 5 | Marcelo | Texas | 555 |
| 6 | Pinto | Delhi | 666 |

| Course_id | Course | Instructor_id | Name | Place | Course_id |
|---|---|---|---|---|---|
| 111 | Science | 1 | Lisa | California | 111 |
| 222 | Finance | 2 | Roy | Bangalore | 222 |
| 333 | History | 3 | Mathew | Mumbai | 333 |
| 444 | Art | 4 | Lorna | Boston | 444 |
| 555 | Music | 5 | Marcelo | Texas | 555 |
| 777 | Physics | NULL | NULL | NULL | NULL |

## Right Join

- The **right join** command displays the rows that have a matching attribute in both the tables along with the remaining rows of the table on the right side of the join.
- For the rows on the right table, for which there is no matching value for the common attribute on left table, the result table displays NULL.

```
SELECT column_name(s)
FROM table1
RIGHT JOIN table2
ON table1.column_name = table2.column_name;
```



جامعة القاهرة الجديدة التكنولوجية NEW CAIRO TECHNOLOGICAL UNIVERSITY

كلية تكنولوجيا الصناعة والطاقة

# Right Join Example

```
SELECT * FROM course RIGHT JOIN instructor ON course. Course_id
= instructor. Course_id;
```

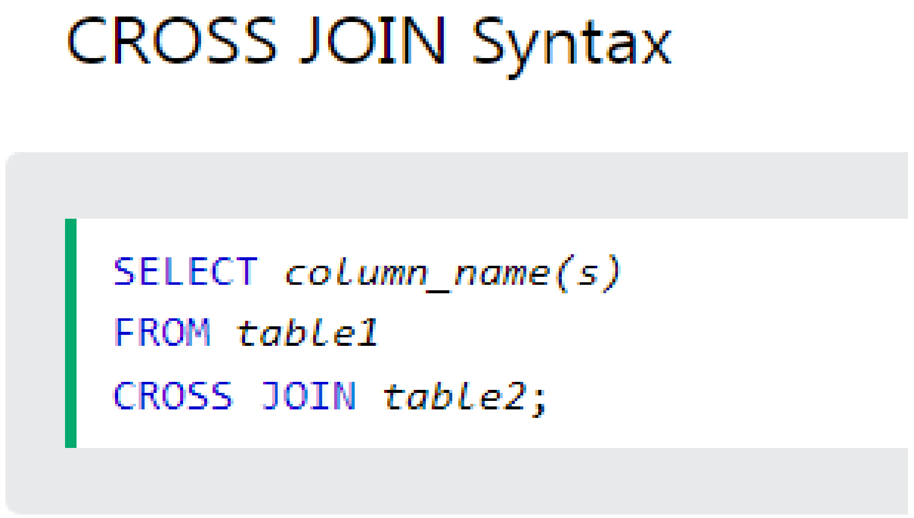
| Course_id | Course |
|---|---|
| 111 | Science |
| 222 | Finance |
| 333 | History |
| 444 | Art |
| 555 | Music |
| 777 | Physics |

| Instructor_id | Name | Place | Course_id |
|---|---|---|---|
| 1 | Lisa | California | 111 |
| 2 | Roy | Bangalore | 222 |
| 3 | Mathew | Mumbai | 333 |
| 4 | Lorna | Boston | 444 |
| 5 | Marcelo | Texas | 555 |
| 6 | Pinto | Delhi | 666 |

| Course_id | Course | Instructor_id | Name | Place | Course_id |
|---|---|---|---|---|---|
| 111 | Science | 1 | Lisa | California | 111 |
| 222 | Finance | 2 | Roy | Bangalore | 222 |
| 333 | History | 3 | Mathew | Mumbai | 333 |
| 444 | Art | 4 | Lorna | Boston | 444 |
| 555 | Music | 5 | Marcelo | Texas | 555 |
| NULL | NULL | 6 | Pinto | Delhi | 666 |

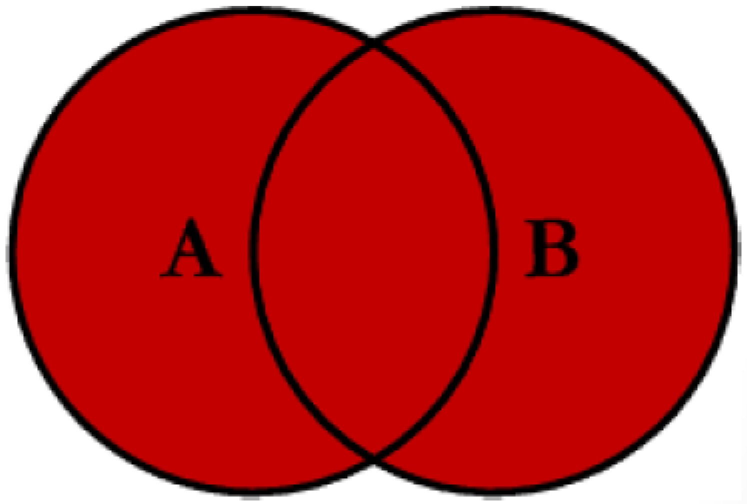
## Full Outer Join

- The **cross join** command displays all rows from both the tables.
- For the rows for which there is no matching value for the common attribute from both the tables, the result table displays NULL.



CROSS JOIN Syntax

```
SELECT column_name(s)
FROM table1
CROSS JOIN table2;
```



# Full Outer Join Example

```
SELECT * FROM course FULL OUTER JOIN instructor ON course.
Course_id = instructor. Course_id;
```

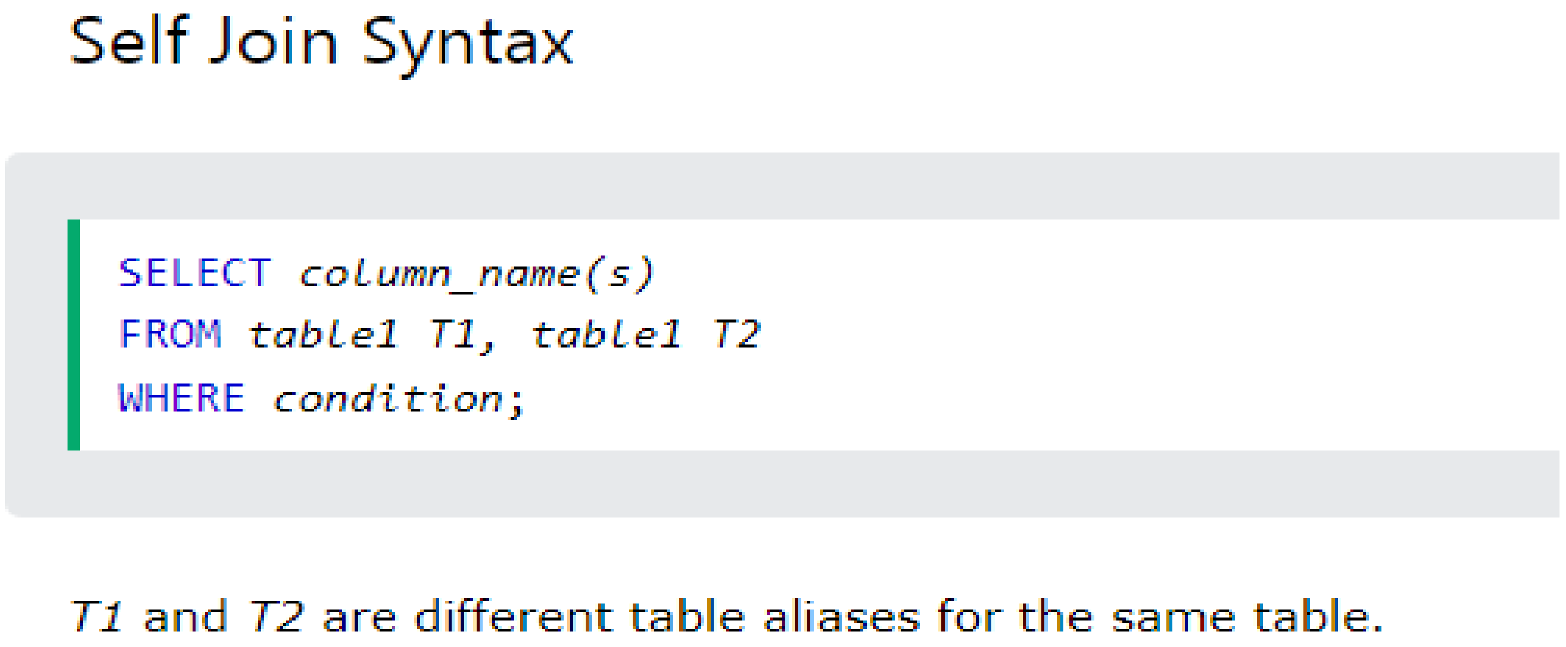
| Course_id | Course |
|---|---|
| 111 | Science |
| 222 | Finance |
| 333 | History |
| 444 | Art |
| 555 | Music |
| 777 | Physics |

| Instructor_id | Name | Place | Course_id |
|---|---|---|---|
| 1 | Lisa | California | 111 |
| 2 | Roy | Bangalore | 222 |
| 3 | Mathew | Mumbai | 333 |
| 4 | Lorna | Boston | 444 |
| 5 | Marcelo | Texas | 555 |
| 6 | Pinto | Delhi | 666 |

| Course_id | Course | Instructor_id | Name | Place | Course_id |
|---|---|---|---|---|---|
| 111 | Science | 1 | Lisa | California | 111 |
| 222 | Finance | 2 | Roy | Bangalore | 222 |
| 333 | History | 3 | Mathew | Mumbai | 333 |
| 444 | Art | 4 | Lorna | Boston | 444 |
| 555 | Music | 5 | Marcelo | Texas | 555 |
| 777 | Physics | NULL | NULL | NULL | NULL |
| NULL | NULL | 6 | Pinto | Delhi | 666 |

## Self Join

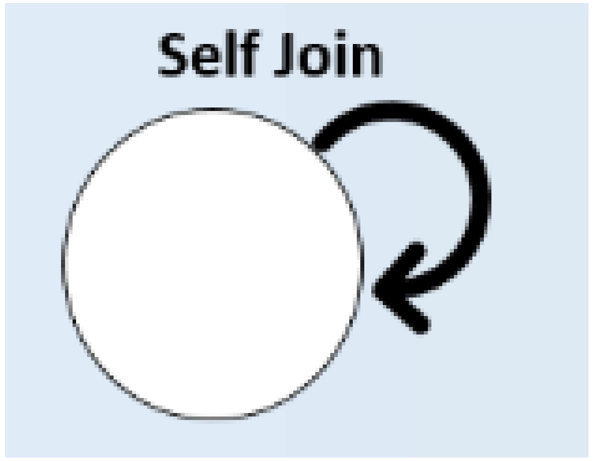
- A self join is a regular join, but the table is joined with itself.



### Self Join Syntax

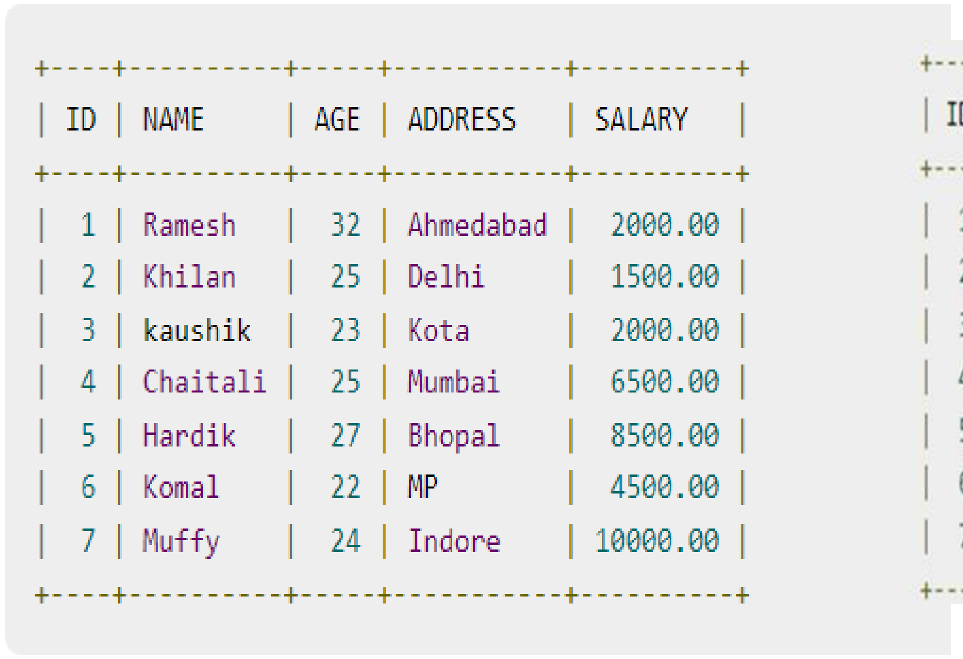
```
SELECT column_name(s)
FROM table1 T1, table1 T2
WHERE condition;
```

*T1* and *T2* are different table aliases for the same table.

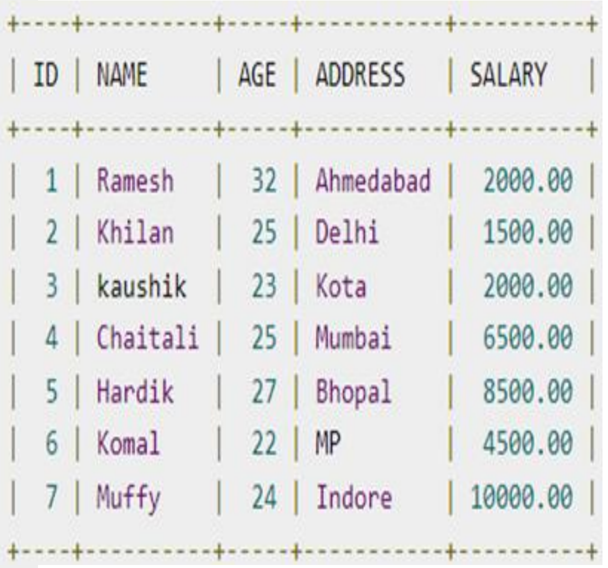


# Self Join Example

**CUSTOMERS Table** is as follows.



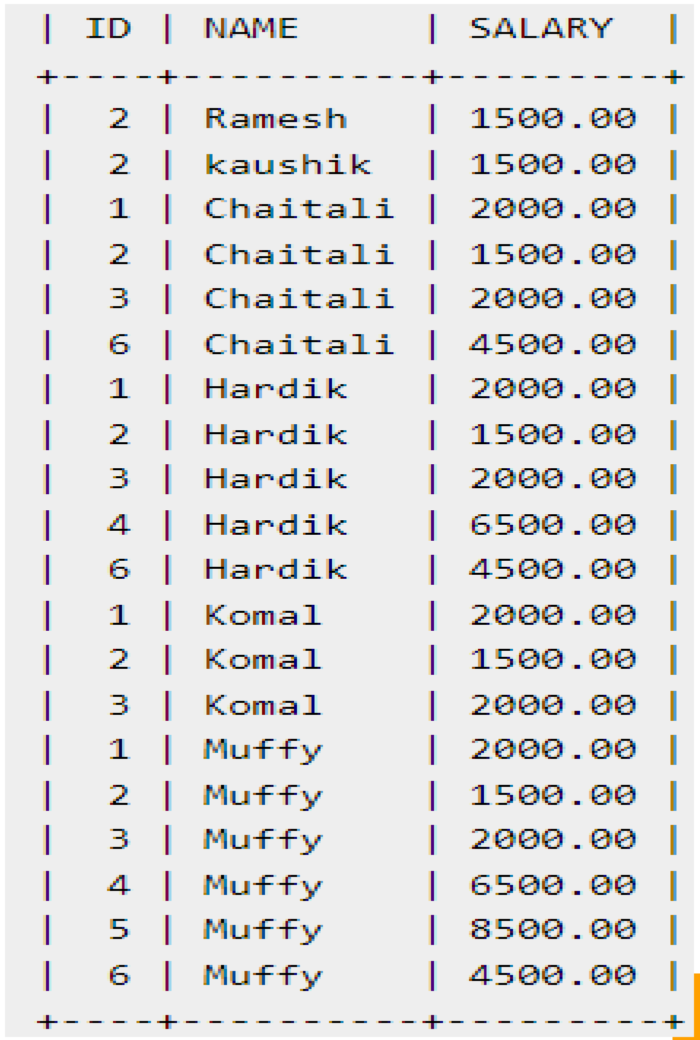
```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```



```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

Now, let us join this table using SELF JOIN as follows –

```
SQL> SELECT  a.ID, b.NAME, a.SALARY
   FROM CUSTOMERS a, CUSTOMERS b
   WHERE a.SALARY < b.SALARY;
```

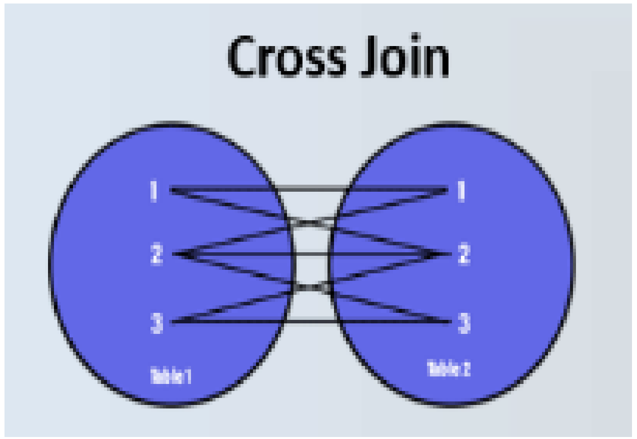


```
| ID  | NAME       | SALARY    |
+-----+------------+-----------+
|   2 | Ramesh     | 1500.00   |
|   2 | kaushik    | 1500.00   |
|   1 | Chaitali   | 2000.00   |
|   2 | Chaitali   | 1500.00   |
|   3 | Chaitali   | 2000.00   |
|   6 | Chaitali   | 4500.00   |
|   1 | Hardik     | 2000.00   |
|   2 | Hardik     | 1500.00   |
|   3 | Hardik     | 2000.00   |
|   4 | Hardik     | 6500.00   |
|   6 | Hardik     | 4500.00   |
|   1 | Komal      | 2000.00   |
|   2 | Komal      | 1500.00   |
|   3 | Komal      | 2000.00   |
|   1 | Muffy      | 2000.00   |
|   2 | Muffy      | 1500.00   |
|   3 | Muffy      | 2000.00   |
|   4 | Muffy      | 6500.00   |
|   5 | Muffy      | 8500.00   |
|   6 | Muffy      | 4500.00   |
+-----+------------+-----------+
```

## Cross (Cartesian) Join

- This join produces a result where the number of rows in the first table gets multiplied with the rows in the second table.
- It returns the Cartesian product of the sets of records from two or more joined tables. This kind of result is called the Cartesian Product.
- If we use the WHERE clause with this join, then this will work as an inner join.

```sql
SELECT column_name(s)
FROM table1
CROSS JOIN table2;
```



# Cross (Cartesian) Join Example

```
SELECT p.product_name, p.prod_unit, c.company_name
FROM product p
CROSS JOIN company c;
```

| Prod_id | Product_name | Prod_unit | Company_id |
|---|---|---|---|
| 1 | Chex mix | Pcs | 12 |
| 2 | Cheez-it | Pcs | 15 |
| 3 | Biscuit | pcs | 16 |

| Company_id | Company_name | Company_city |
|---|---|---|
| 15 | Foodies | Delhi |
| 16 | Jack n Jill | Cuttack |
| 17 | Natural | Bangalore |

| p.product_name | p.prod_unit | c.company_name |
|---|---|---|
| Chex mix | Pcs | Foodies |
| Cheez-it | Pcs | Foodies |
| Biscuit | Pcs | Foodies |
| Chex mix | Pcs | Jack n Jill |
| Cheez-it | Pcs | Jack n Jill |
| Biscuit | Pcs | Jack n Jill |
| Chex mix | Pcs | Natural |
| Cheez-it | Pcs | Natural |
| Biscuit | Pcs | Natural |

كلية تكنولوجيا الصناعة والطاقة

جامعة القاهرة الجديدة التكنولوجية
NCT
NEW CAIRO TECHNOLOGICAL UNIVERSITY

## Thank you