DATA ANALYSIS TRAINING WITH SQL

Tech365
Leveraging technology for growth

www.tech365.ng/training

OUTLINE

Section 1: Introduction to SQL

- What is SQL
- SQL Syntax
- SQL Sample Database

Section 2: Querying Data

- SELECT Statement

Section 3: Sorting Data

- ORDER BY Clause

Section 4: Filtering Data

- DISTINCT
- LIMIT
- WHERE Clause
- Comparison operators
- Logical operators
- AND operator
- OR operator
- BETWEEN Operator
- IN Operator
- LIKE Operator
- IS NULL Operator
- NOT operator

OUTLINE contd..

Section 5: Joining Multiple Tables

- SQL Aliases
- INNER JOIN
- LEFT OUTER JOIN
- FULL OUTER JOIN

- **Section 6: Aggregate Functions**
- AVG
- COUNT
- SUM
- MAX
- MIN

OUTLINE contd..

- **Section 7: Grouping Data**
- GROUP BY
- HAVING
- GROUPING SETS

- **Section 8: SET Operators**
- UNION and UNION ALL
- INTERSECT
- MINUS

- **Section 9. Subquery**
- Subquery

- **Section 10: Modifying data**
- INSERT
- UPDATE
- DELETE

- **Section 11: Working with table structures**
- CREATE TABLE
- ALTER TABLE
- DROP TABLE
- TRUNCATE TABLE

OUTLINE contd..

- **Section 12: Constraints**
- PRIMARY KEY
- FOREIGN KEY
- UNIQUE
- NOT NULL

INTRODUCTION

What is SQL

- SQL is a **programming language** designed to **manage data** stored in a relational database management system (RDBMS).
- SQL stands for the structured query language. It is pronounced as /ˈɛs kjuː ˈɛl/ or /ˈsiːkwəl/.
- SQL consists of a data definition language, data manipulation language, and a data control language.
- The data definition language deals with the schema creation and modification e.g., CREATE TABLE statement allows you to create a new table in the database and the ALTER TABLE statement changes the structure of an existing table.
- The data manipulation language provides the constructs to query data such as the SELECT statement and to update the data such as INSERT, UPDATE, and DELETE statements.
- The data control language consists of the statements that deal with the user authorization and security such as GRANT and REVOKE statements.

SQL Standard

- SQL was one of the first commercial database languages since 1970. Since then different database vendors implemented SQL in their products with some variations. To bring greater conformity between the vendors, the American Standards Institute (ANSI) published the first SQL standard in 1986.
- ANSI then updated the SQL standard in 1992, known as SQL92 and SQL2, and again in 1999 as SQL99 and SQL3. Every time, ANSI added new features and commands into the SQL language.
- The SQL Standard is now maintained by both ANSI and International Standards Organization as ISO/IEC 9075 standard. The latest release standard is SQL:2011.
- The SQL standard formalizes SQL syntax structures and behaviors across database products. It becomes even more important to the open-source databases such as MySQL and PostgreSQL where the RDBMS are developed mainly by the communities rather than big corporations.

SQL Dialet

- The community constantly requests new features and capabilities that do not exist in the SQL standard yet, therefore, even with the SQL standard in place, there are many SQL dialects in various database products.
- Because ANSI and ISO have not yet developed these important features, RDBMS vendors (or communities) are free to invent their own new syntax structure.
- The following are the most popular dialects of SQL:
- PL/SQL stands for procedural language/SQL. It is developed by Oracle for the Oracle Database.
- Transact-SQL or T-SQL is developed by Microsoft for Microsoft SQL Server.
- PL/pgSQL stands for Procedural Language/PostgreSQL that consists of SQL dialect and extensions implemented in PostgreSQL
- MySQL has its own procedural language since version 5. Note that MySQL was acquired by Oracle.
- In each tutorial, we will explain the SQL syntax structures and behaviors that are valid across the databases. We also will discuss the exceptions if they exist in a particular database.

SQL Syntax

SQL Syntax

- SQL is a declarative language, therefore, its syntax reads like a natural language. An SQL statement begins with a verb that describes the action, for example, SELECT, INSERT, UPDATE or DELETE. Following the verb are the subject and predicate.

- A predicate specifies conditions that can be evaluated as true, false, or unknown.

SQL Syntax

- See the following SQL statement:

```
SELECT
    first_name
FROM
    employees
WHERE
    YEAR(hire_date) = 2000;
```

- Get the first names of employees who were hired in 2000.

SQL commands

- SQL is made up of many commands.
- Each SQL command is typically terminated with a semicolon (;).
- For example, the following are two different SQL commands separated by a semicolon (;).

```
SELECT
    first_name, last_name
FROM
    employees;

DELETE FROM employees
WHERE
    hire_date < '1990-01-01';
```

Literals

- Literals are explicit values which are also known as constants. SQL provides three kinds of literals: string, numeric, and binary.
- **String literal** consists of one or more alphanumeric characters surrounded by **single quotes**
- SQL is **case sensitive** with respect to string literals, so the value 'John' is not the same as 'JOHN'.
- Numeric literals are the integer, decimal, or scientific notation

```
'John'
'1990-01-01'
'50'
```

```
200
-5
6.0221415E23
```

```
x'01'
x'0f0ff'
```

Keywords

- SQL has many keywords that have special meanings such as SELECT, INSERT, UPDATE, DELETE, and DROP. These keywords are the reserved words, therefore, you cannot use them as the name of tables, columns, indexes, views, stored procedures, triggers, or other database objects.

Identifiers

- Identifiers refer to specific objects in the database such as tables, columns, indexes, etc. SQL is case-insensitive with respect to keywords and identifiers.

- To make the SQL commands more readable and clear, we will use the SQL keywords in uppercase and identifiers in lower case throughout the tutorials.

```
Select  * From employees;

SELECT * FROM EMPLOYEES;

select * from employees;

SELECT * FROM employees;
```

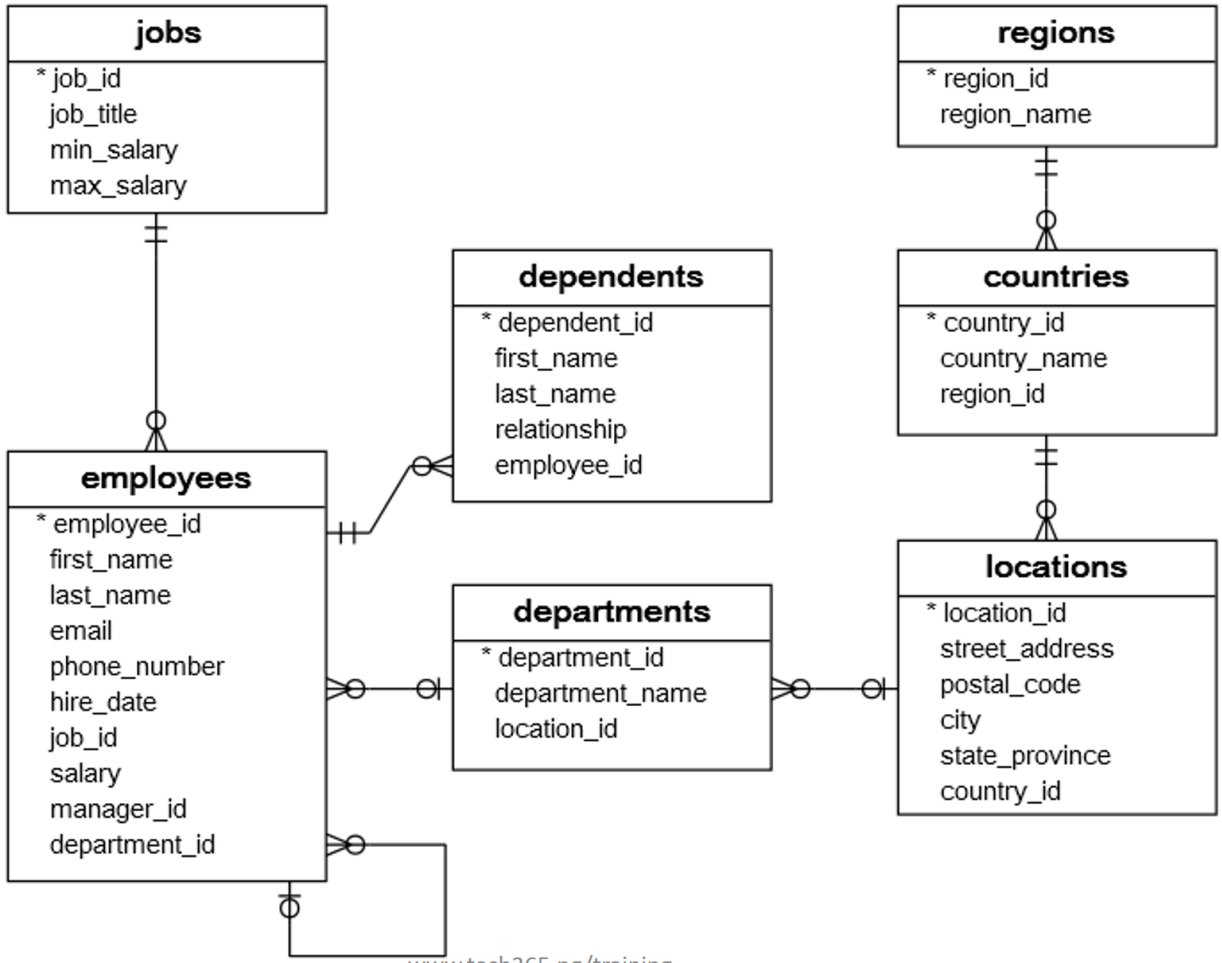
Comments

- To document SQL statements, you use the SQL comments. When parsing SQL statements with comments, the database engine ignores the characters in the comments.

- A comment is denoted by two consecutive hyphens (--) that allow you to comment the remaining line.

```
SELECT
    employee_id, salary
FROM
    employees
WHERE
    salary < 3000;-- employees with low salary
```

Sample Database


jobs
* job_id
job_title
min_salary
max_salary
regions
* region_id
region_name
dependents
* dependent_id
first_name
last_name
relationship
employee_id
countries
* country_id
country_name
region_id
employees
* employee_id
first_name
last_name
email
phone_number
hire_date
job_id
salary
manager_id
department_id
departments
* department_id
department_name
location_id
locations
* location_id
street_address
postal_code
city
state_province
country_id

The HR sample database has seven tables:

- The **employees** table stores the data of employees.
- The **jobs** table stores the job data including job title and salary range.
- The **departments** table stores department data.
- The **dependents** table stores the employee's dependents.
- The **locations** table stores the location of the departments of the company.
- The **countries** table stores the data of countries where the company is doing business.
- The **regions** table stores the data of regions such as Asia, Europe, America, and the Middle East and Africa. The countries are grouped into regions.

Querying Data

SQL SELECT

- The SQL SELECT statement selects data from one or more tables. The following shows the basic syntax of the SELECT statement that selects data from a single table.

```
SELECT
    select_list
FROM
    table_name;
```

1. SQL SELECT – selecting data from all columns example

```
SELECT * FROM employees;
```

Query

```
+-------------+------------+------------+-----------------------------------
| employee_id | first_name | last_name  | email
+-------------+------------+------------+-----------------------------------
|         100 | Steven     | King       | steven.king@sqltutorial.org
|         101 | Neena      | Kochhar    | neena.kochhar@sqltutorial.org
|         102 | Lex        | De Haan    | lex.de haan@sqltutorial.org
|         103 | Alexander  | Hunold     | alexander.hunold@sqltutorial.org
|         104 | Bruce      | Ernst      | bruce.ernst@sqltutorial.org
|         105 | David      | Austin     | david.austin@sqltutorial.org
|         106 | Valli      | Pataballa  | valli.pataballa@sqltutorial.org
|         107 | Diana      | Lorentz    | diana.lorentz@sqltutorial.org
|         108 | Nancy      | Greenberg  | nancy.greenberg@sqltutorial.org
...
```

Result

2. SQL SELECT – selecting data from specific columns

```
SELECT
    employee_id,
    first_name,
    last_name,
    hire_date
FROM
    employees;
```

```
+-------------+-------------+-------------+------------+
| employee_id | first_name  | last_name   | hire_date  |
+-------------+-------------+-------------+------------+
|         100 | Steven      | King        | 1987-06-17 |
|         101 | Neena       | Kochhar     | 1989-09-21 |
|         102 | Lex         | De Haan     | 1993-01-13 |
|         103 | Alexander   | Hunold      | 1990-01-03 |
|         104 | Bruce       | Ernst       | 1991-05-21 |
|         105 | David       | Austin      | 1997-06-25 |
|         106 | Valli       | Pataballa   | 1998-02-05 |
|         107 | Diana       | Lorentz     | 1999-02-07 |
|         108 | Nancy       | Greenberg   | 1994-08-17 |
|         109 | Daniel      | Faviet      | 1994-08-16 |
|         110 | John        | Chen        | 1997-09-28 |
...
```

3. SELECT – performing a simple calculation

```
SELECT
    first_name,
    last_name,
    salary,
    salary * 1.05
FROM
    employees;
```

```
+------------+------------+---------+---------------+
| first_name | last_name  | salary  | salary * 1.05 |
+------------+------------+---------+---------------+
| Steven     | King       | 24000.00 |    25200.0000 |
| Neena      | Kochhar    | 17000.00 |    17850.0000 |
| Lex        | De Haan    | 17000.00 |    17850.0000 |
| Alexander  | Hunold     |  9000.00 |     9450.0000 |
| Bruce      | Ernst      |  6000.00 |     6300.0000 |
| David      | Austin     |  4800.00 |     5040.0000 |
| Valli      | Pataballa  |  4800.00 |     5040.0000 |
| Diana      | Lorentz    |  4200.00 |     4410.0000 |
| Nancy      | Greenberg  | 12000.00 |    12600.0000 |
...
```

Using Alias

- For example, the following SELECT statement uses the new_salary as the column alias for the salary * 1.05 expression:

```
expression AS column_alias
```

```
SELECT
    first_name,
    last_name,
    salary,
    salary * 1.05 AS new_salary
FROM
    employees;
```

Sorting Data

ORDER BY clause

- The ORDER BY is an optional clause of the SELECT statement. The ORDER BY clause allows you to sort the rows returned by the SELECT clause by one or more sort expressions in ascending or descending order.

```
SELECT
        employee_id,
        first_name,
        last_name,
        hire_date,
        salary
FROM
        employees
ORDER BY
        hire_date DESC;
```

Filtering Data

DISTINCT operator

- To remove duplicate rows from a result set, you use the DISTINCT operator in the SELECT clause

```
SELECT
    DISTINCT salary
FROM
    employees
ORDER BY salary DESC;
```

LIMIT clause

- To limit the number of rows returned by a select statement, you use the LIMIT and OFFSET clauses.

```
SELECT
    employee_id,
    first_name,
    last_name
FROM
    employees
ORDER BY
        first_name
LIMIT 5;
```

WHERE clause

- To select specific rows from a table, you use a WHERE clause in the SELECT statement.

```
SELECT
        employee_id,
        first_name,
        last_name,
        salary
FROM
        employees
WHERE
        salary > 14000
ORDER BY
        salary DESC;
```

Tech365
Leveraging technology for growth

Comparison Operators

Operator	Meaning
=	Equal
<>	Not equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to

```
SELECT
    employee_id, first_name, last_name, salary
FROM
    employees
WHERE
    salary > 10000 AND department_id = 8
ORDER BY salary DESC;
```

Logical Operators

```sql
SELECT
    first_name, last_name, salary
FROM
    employees
WHERE
    salary > 5000 AND salary < 7000
ORDER BY salary;
```

```sql
SELECT
    first_name, last_name, salary
FROM
    employees
WHERE
    salary = 7000 OR salary = 8000
ORDER BY salary;
```

Operator	Meaning
ALL	Return true if all comparisons are true
AND	Return true if both expressions are true
ANY	Return true if any one of the comparisons is true.
BETWEEN	Return true if the operand is within a range
EXISTS	Return true if a subquery contains any rows
IN	Return true if the operand is equal to one of the value in a list
LIKE	Return true if the operand matches a pattern
NOT	Reverse the result of any other Boolean operator.
OR	Return true if either expression is true
SOME	Return true if some of the expressions are true

Logical operator

```sql
SELECT
    first_name, last_name, phone_number
FROM
    employees
WHERE
    phone_number IS NULL
ORDER BY first_name , last_name;
```

```sql
SELECT
    first_name, last_name, salary
FROM
    employees
WHERE
    salary BETWEEN 9000 AND 12000
ORDER BY salary;
```

```sql
SELECT
    first_name, last_name, department_id
FROM
    employees
WHERE
    department_id IN (8, 9)
ORDER BY department_id;
```

Tech365

LIKE operator

- The LIKE operator compares a value to similar values using a wildcard operator. SQL provides two wildcards used in conjunction with the LIKE operator:

- The percent sign (%) represents zero, one, or multiple characters.
- The underscore sign (_) represents a single character.

```
SELECT
    employee_id, first_name, last_name
FROM
    employees
WHERE
    first_name LIKE 'jo%'
ORDER BY first_name;
```

```
SELECT
    employee_id, first_name, last_name
FROM
    employees
WHERE
    first_name LIKE '_h%'
ORDER BY first_name;
```

ALL operator

- The ALL operator compares a value to all values in another value set. The ALL operator must be preceded by a comparison operator and followed by a subquery.

```
SELECT
    first_name, last_name, salary
FROM
    employees
WHERE
    salary >= ALL (SELECT
            salary
        FROM
            employees
        WHERE
            department_id = 8)
ORDER BY salary DESC;
```

ANY operator

- The ANY operator compares a value to any value in a set according to the condition

- Similar to the ALL operator, the ANY operator must be preceded by a comparison operator and followed by a subquery.

```
SELECT
    first_name, last_name, salary
FROM
    employees
WHERE
    salary > ANY(SELECT
            AVG(salary)
        FROM
            employees
        GROUP BY department_id)
ORDER BY first_name , last_name;
```

Tech365

EXISTS operator

- The EXISTS operator tests if a subquery contains any rows
- If the subquery returns one or more rows, the result of the EXISTS is true; otherwise, the result is false.

```
SELECT
    first_name, last_name
FROM
    employees e
WHERE
    EXISTS( SELECT
            1
        FROM
            dependents d
        WHERE
            d.employee_id = e.employee_id);
```

Tech365

NULL

- NULL is special in SQL. NULL indicates that the data is unknown, inapplicable, or even does not exist. In other words, NULL represents the missing data in the database.

- For example, if employees do not have phone numbers, you can store their phone numbers as empty strings.

- However, if you don’t know their phone numbers when you save the employee records, you need to use the NULL for the unknown phone numbers.

```
SELECT
        employee_id,
        first_name,
        last_name,
        phone_number
FROM
        employees
WHERE
        phone_number IS NULL;
```

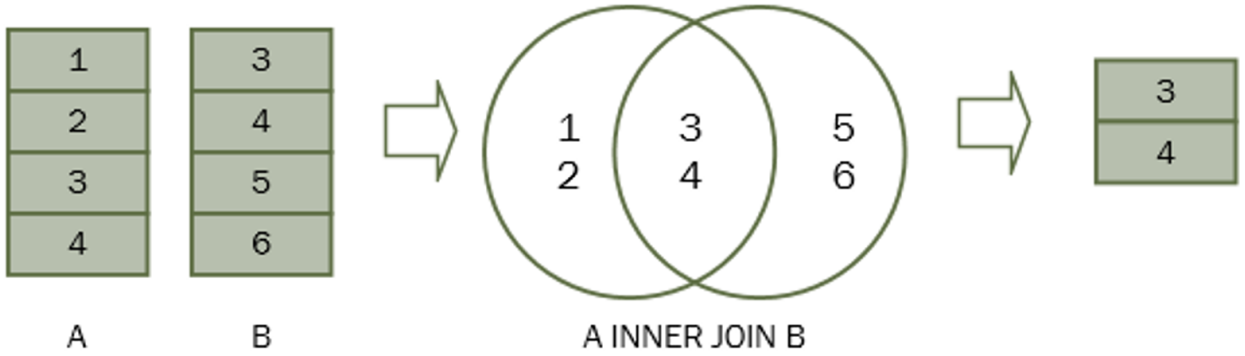
Tech365

Join

JOIN

- The process of linking tables is called joining. SQL provides many kinds of joins such as inner join, left join, right join, full outer join, etc.
- Suppose, you have two tables: A and B.
- Table A has four rows: (1,2,3,4) and table B has four rows: (3,4,5,6)
- When table A joins with table B using the inner join, you have the result set (3,4) that is the intersection of table A and table B.

INNER JOIN



INNER JOIN SYNTAX

```
SELECT a
FROM A
INNER JOIN B ON b = a;
```

```
SELECT
  A.n
FROM A
INNER JOIN B ON B.n = A.n
INNER JOIN C ON C.n = A.n;
```

Tech365
Leveraging technology for growth

INNER JOIN contd...

- Each employee belongs to one and only one department while each department can have more than one employee. The relationship between the departments and employees is one-to-many.

- The department_id column in the employees table is the foreign key column that links the employees to the departments table.

- To get the information of the department id 1,2, and 3, you use the following statement.

Before we use inner join,
querying the department table for those in department 1,2 and 3

```
SELECT
        department_id,
        department_name
FROM
        departments
WHERE
        department_id IN (1, 2, 3);
```

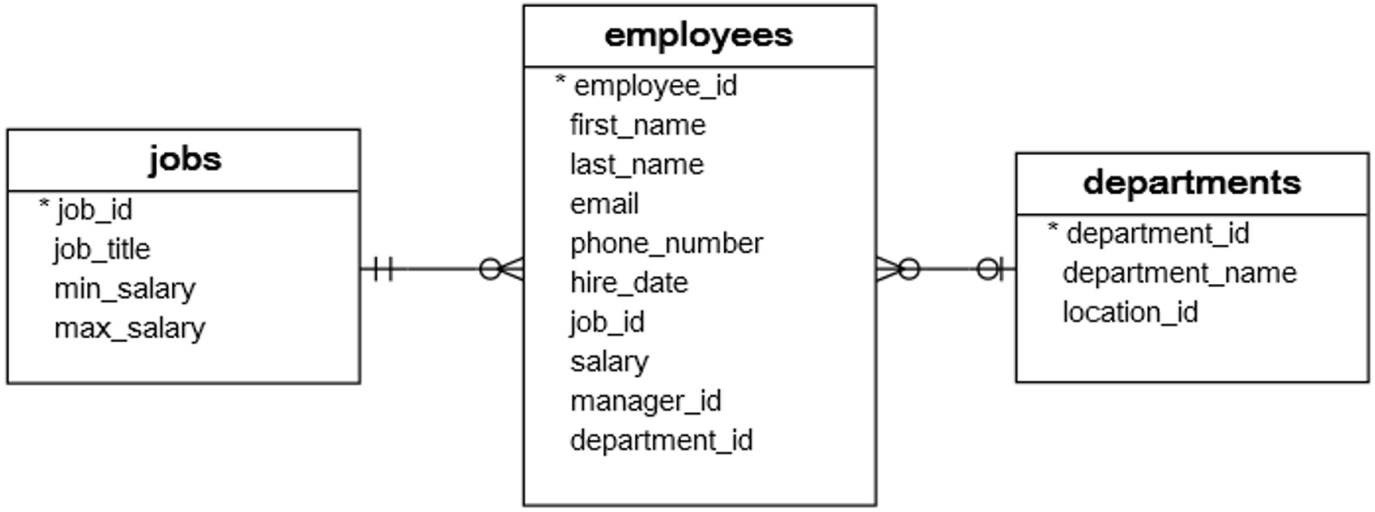
INNER JOIN contd...

To get the information of employees who work in the department id 1, 2 and 3

```
SELECT
    first_name,
    last_name,
    employees.department_id,
    departments.department_id,
    department_name
FROM
    employees
        INNER JOIN
    departments ON departments.department_id = employees.department_id
WHERE
    employees.department_id IN (1 , 2, 3);
```

first_name	last_name	department_id	department_id	department_name
Jennifer	Whalen	1	1	Administration
Michael	Hartstein	2	2	Marketing
Pat	Fay	2	2	Marketing
Den	Raphaely	3	3	Purchasing
Alexander	Khoo	3	3	Purchasing
Shelli	Baida	3	3	Purchasing
Sigal	Tobias	3	3	Purchasing
Guy	Himuro	3	3	Purchasing
Karen	Colmenares	3	3	Purchasing

INNER JOIN contd...



EXERCISE

- Get the first name, last name, job title, and department name of employees who work in department id 1, 2, and 3.

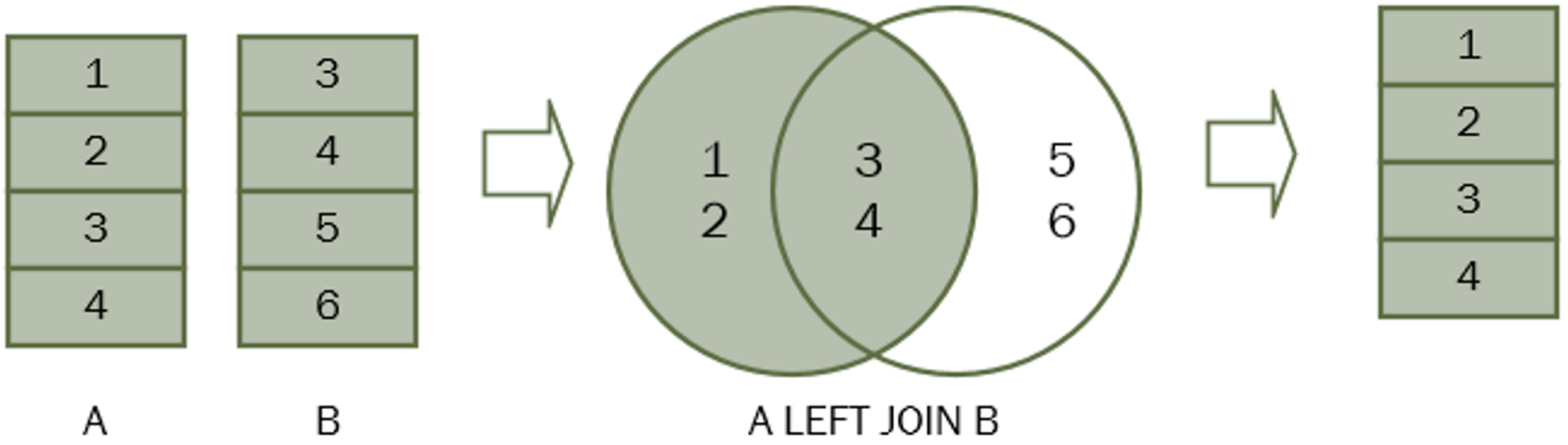
```
SELECT
        first_name,
        last_name,
        job_title,
        department_name
FROM
        employees e
INNER JOIN departments d ON d.department_id = e.department_id
INNER JOIN jobs j ON j.job_id = e.job_id
WHERE
        e.department_id IN (1, 2, 3);
```

LEFT JOIN

- The left join, however, returns all rows from the left table whether or not there is a matching row in the right table.

- Suppose we have two tables A and B. The table A has four rows 1, 2, 3 and 4. The table B also has four rows 3, 4, 5, 6.

- When we join table A with table B, all the rows in table A (the left table) are included in the result set whether there is a matching row in the table B or not.

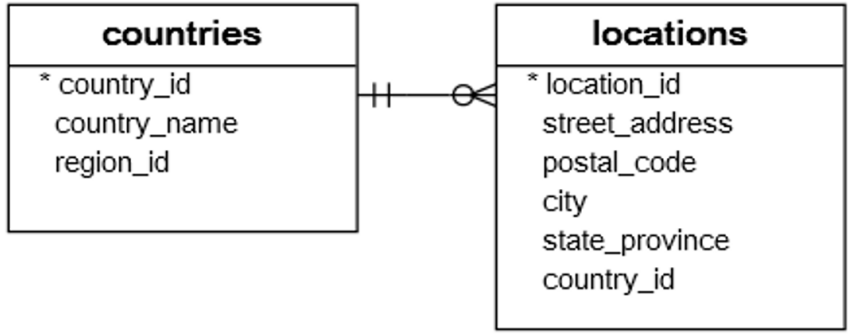

1
2
3
4
3
4
5
6
1
2
3
4
5
6
1
2
3
4
A
B
A LEFT JOIN B

LEFT JOIN SYNTAX

Each location belongs to one and only one country while each country can have zero or more locations.

The relationship between the countries and locations tables is one-to-many.

The country_id column in the locations table is the foreign key that links to the country_id column in the countries table.




```
SELECT
        A.n
FROM
        A
LEFT JOIN B ON B.n = A.n;
```

LEFT JOIN contd...

The following query retrieves the locations located in the US, UK and China:

```
SELECT
        country_id,
        street_address,
        city
FROM
        locations
WHERE
        country_id IN ('US', 'UK', 'CN');
```

we use the LEFT JOIN clause to join the countries table with the locations table as the following query:

```
SELECT
        c.country_name,
        c.country_id,
        l.country_id,
        l.street_address,
        l.city
FROM
        countries c
LEFT JOIN locations l ON l.country_id = c.country_id
WHERE
        c.country_id IN ('US', 'UK', 'CN')
```

Exercises

- find the country that does not have any locations in the locations table

```
SELECT
        country_name
FROM
        countries c
LEFT JOIN locations l ON l.country_id = c.country_id
WHERE
        l.location_id IS NULL
ORDER BY
        country_name;
```

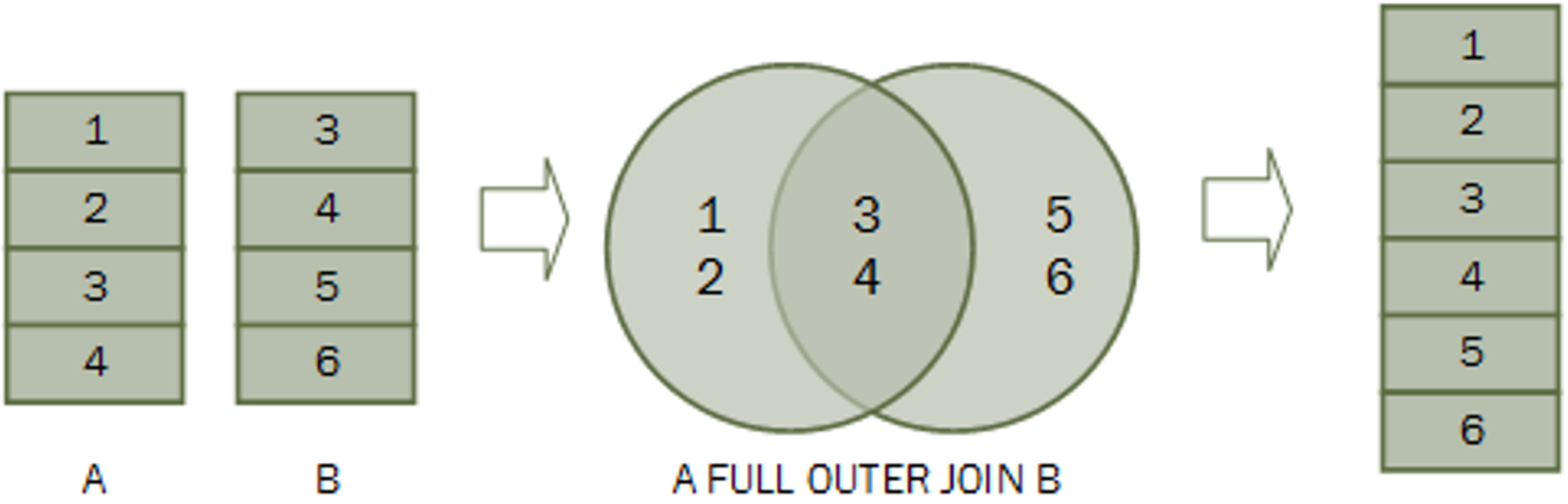
SQL FULL OUTER JOIN

- In theory, a full outer join is the combination of a left join and a right join. The full outer join includes all rows from the joined tables whether or not the other table has the matching row.

- If the rows in the joined tables do not match, the result set of the full outer join contains NULL values for every column of the table that lacks a matching row. For the matching rows, a single row that has the columns populated from the joined table is included in the result set.

```
SELECT column_list
FROM A
FULL OUTER JOIN B ON B.n = A.n;
```

FULL OUTER JOIN contd...



Example of full outer join

- First, create two new tables: baskets and fruits for the demonstration. Each basket stores zero or more fruits and each fruit can be stored in zero or one basket.

```
CREATE TABLE fruits (
        fruit_id INTEGER PRIMARY KEY,
        fruit_name VARCHAR (255) NOT NULL,
        basket_id INTEGER
);
```

```
CREATE TABLE baskets (
        basket_id INTEGER PRIMARY KEY,
        basket_name VARCHAR (255) NOT NULL
);
```

Tech365

Second, insert some sample data into the baskets and fruits tables.

```
INSERT INTO baskets (basket_id, basket_name)
VALUES
        (1, 'A'),
        (2, 'B'),
        (3, 'C');
```

```
INSERT INTO fruits (
        fruit_id,
        fruit_name,
        basket_id
)
VALUES
        (1, 'Apple', 1),
        (2, 'Orange', 1),
        (3, 'Banana', 2),
        (4, 'Strawberry', NULL);
```

Tech365

Third, the following query returns each fruit that is in a basket and each basket that has a fruit, but also returns each fruit that is not in any basket and each basket that does not have any fruit.

```
SELECT
        basket_name,
        fruit_name
FROM
        fruits
FULL OUTER JOIN baskets ON baskets.basket_id = fruits.basket_id;
```

```
 basket_name | fruit_name
-------------+------------
 A           | Apple
 A           | Orange
 B           | Banana
 (null)      | Strawberry
 C           | (null)
```

To find the empty basket, which does not store any fruit

```
SELECT
        basket_name,
        fruit_name
FROM
        fruits
FULL OUTER JOIN baskets ON baskets.basket_id = fruits.basket_id
WHERE
        fruit_name IS NULL;
```

```
 basket_name | fruit_name
-------------+------------
 C           | (null)
(1 row)
```

Exercise: which fruit is not in any basket?

```
SELECT
        basket_name,
        fruit_name
FROM
        fruits
FULL OUTER JOIN baskets ON baskets.basket_id = fruits.basket_id
WHERE
        basket_name IS NULL;
```

```
 basket_name | fruit_name
-------------+------------
(null)       | Strawberry
(1 row)
```

Group By

GROUP BY

- The GROUP BY is an optional clause of the SELECT statement. The GROUP BY clause allows you to group rows based on values of one or more columns. It returns one row for each group.

```
SELECT
        column1,
        column2,
        aggregate_function(column3)
FROM
        table_name
GROUP BY
        column1,
        column2;
```

GROUP BY contd...

- The following picture illustrates shows how the GROUP BY clause works:

id	fruit
1	Apple
2	Orange
3	Apple
4	Banana
5	Orange

fruit
Apple
Banana
Orange

GROUP BY contd...

- In practice, you often use the GROUP BY clause with an aggregate function such as MIN, MAX, AVG, SUM, or COUNT to calculate a measure that provides the information for each group.
- For example, the following illustrates how the GROUP BY clause works with the COUNT aggregate function

id	fruit
1	Apple
2	Orange
3	Apple
4	Banana
5	Orange

fruit	count(id)
Apple	2
Banana	1
Orange	2

The following example uses the GROUP BY clause to group the values in department_id column of the employees table

Query

```
SELECT
    department_id
FROM
    employees
GROUP BY
    department_id;
```

Result

```
+---------------+
| department_id |
+---------------+
|             1 |
|             2 |
|             3 |
|             4 |
|             5 |
|             6 |
|             7 |
|             8 |
|             9 |
|            10 |
|            11 |
+---------------+
11 rows in set (0.00 sec)
```

Using Distinct

- The department_id column of the employees table has 40 rows, including duplicate department_id values. However, the GROUP BY groups these values into groups.

- Without an aggregate function, the GROUP BY behaves like the DISTINCT keyword

```
SELECT
   DISTINCT department_id
FROM
   employees
ORDER BY
   department_id;
```

Using group by with Agregate function

```
SELECT
        department_id,
        COUNT(employee_id) headcount
FROM
        employees
GROUP BY
        department_id;
```

SQL GROUP BY with INNER JOIN

```
SELECT
        department_name,
        COUNT(employee_id) headcount
FROM
        employees e
INNER JOIN departments d ON d.department_id = e.department_id
GROUP BY
        department_name;
```

Tech365
Leveraging technology for growth

SQL GROUP BY with ORDER BY

```
SELECT
    department_name,
    COUNT(employee_id) headcount
FROM
    employees e
        INNER JOIN
    departments d ON d.department_id = e.department_id
GROUP BY department_name
ORDER BY headcount DESC;
```

SQL GROUP BY with HAVING

```
SELECT
    department_name,
    COUNT(employee_id) headcount
FROM
    employees e
        INNER JOIN
    departments d ON d.department_id = e.department_id
GROUP BY department_name
HAVING headcount > 5
ORDER BY headcount DESC;
```

SQL GROUP BY with MIN, MAX, and AVG

```
SELECT
    department_name,
    MIN(salary) min_salary,
    MAX(salary) max_salary,
    ROUND(AVG(salary), 2) average_salary
FROM
    employees e
        INNER JOIN
    departments d ON d.department_id = e.department_id
GROUP BY
    department_name;
```

SQL GROUP BY with SUM function

```
SELECT
    department_name,
    SUM(salary) total_salary
FROM
    employees e
        INNER JOIN
    departments d ON d.department_id = e.department_id
GROUP BY
    department_name;
```

SQL GROUP BY multiple columns

```
SELECT
    department_name,
    job_title,
    COUNT(employee_id)
FROM
    employees e
        INNER JOIN
    departments d ON d.department_id = e.department_id
        INNER JOIN
    jobs j ON j.job_id = e.job_id
GROUP BY department_name ,
         job_title;
```

Having

SQL HAVING clause

- The HAVING clause is often used with the GROUP BY clause in the SELECT statement. If you use a HAVING clause without a GROUP BY clause, the HAVING clause behaves like the WHERE clause.

- The WHERE clause applies the condition to individual rows before the rows are summarized into groups by the GROUP BY clause. However, the HAVING clause applies the condition to the groups after the rows are grouped into groups.

- Therefore, it is important to note that the HAVING clause is applied after whereas the WHERE clause is applied before the GROUP BY clause.

WHERE example

- To get the managers and their direct reports, you use the GROUP BY clause to group employees by the managers and use the COUNT function to count the direct reports.

```
SELECT
    manager_id,
    first_name,
    last_name,
    COUNT(employee_id) direct_reports
FROM
    employees
WHERE
    manager_id IS NOT NULL
GROUP BY manager_id;
```

Tech365

HAVING example

- To find the managers who have at least five direct reports, you add a HAVING clause to the query above as the following:

```
SELECT
    manager_id,
    first_name,
    last_name,
    COUNT(employee_id) direct_reports
FROM
    employees
WHERE
    manager_id IS NOT NULL
GROUP BY manager_id
HAVING direct_reports >= 5;
```

SQL HAVING with SUM

- The following statement calculates the sum of salary that the company pays for each department and selects only the departments with the sum of salary between 20000 and 30000.

```
SELECT
    department_id, SUM(salary)
FROM
    employees
GROUP BY department_id
HAVING SUM(salary) BETWEEN 20000 AND 30000
ORDER BY SUM(salary);
```

SQL HAVING with MIN function

- To find the department that has employees with the lowest salary greater than 10000, you use the following query

```
SELECT
        e.department_id,
        department_name,
        MIN(salary)
FROM
        employees e
INNER JOIN departments d ON d.department_id = e.department_id
GROUP BY
        e.department_id
HAVING
        MIN(salary) >= 10000
ORDER BY
        MIN(salary);
```

SQL HAVING clause with AVG

- To find the departments that have the average salaries of employees between 5000 and 7000, you use the AVG function as the following query

```
SELECT
        e.department_id,
        department_name,
        ROUND(AVG(salary), 2)
FROM
        employees e
INNER JOIN departments d ON d.department_id = e.department_id
GROUP BY
        e.department_id
HAVING
        AVG(salary) BETWEEN 5000
AND 7000
ORDER BY
        AVG(salary);
```

Tech365

Case

SQL CASE

- The SQL CASE expression allows you to evaluate a list of conditions and returns one of the possible results. The CASE expression has two formats: simple CASE and searched CASE.

- You can use the CASE expression in a clause or statement that allows a valid expression. For example, you can use the CASE expression in statements such as SELECT, DELETE, and UPDATE or in clauses such as SELECT, ORDER BY, and HAVING.

```
CASE expression
WHEN when_expression_1 THEN
        result_1
WHEN when_expression_2 THEN
        result_2
WHEN when_expression_3 THEN
        result_3
...
ELSE
        else_result
END
```

Simple SQL CASE

- Suppose the current year is 2000
- We can use the simple CASE expression to get the work anniversaries of employees

```
SELECT
    first_name,
    last_name,
    hire_date,
    CASE (2000 - YEAR(hire_date))
        WHEN 1 THEN '1 year'
        WHEN 3 THEN '3 years'
        WHEN 5 THEN '5 years'
        WHEN 10 THEN '10 years'
        WHEN 15 THEN '15 years'
        WHEN 20 THEN '20 years'
        WHEN 25 THEN '25 years'
        WHEN 30 THEN '30 years'
    END aniversary
FROM
    employees
ORDER BY first_name;
```

Search CASE expression

```
SELECT
    first_name,
    last_name,
    CASE
        WHEN salary < 3000 THEN 'Low'
        WHEN salary >= 3000 AND salary <= 5000 THEN 'Average'
        WHEN salary > 5000 THEN 'High'
    END evaluation
FROM
    employees;
```

Aggregate Function

SQL AVG

- The SQL AVG function is an aggregate function that calculates the average value of a set.
- To calculate the average salary of all employees, you apply the AVG function to the salary column

```
SELECT
    AVG(salary)
FROM
    employees;
```

```
SELECT
    AVG(salary)
FROM
    employees
WHERE
    job_id = 6;
```

Tech365

Exercises: Calculate the departments and the average salary of employees of each department.

```
SELECT
        department_id,
        AVG(salary)
FROM
        employees
GROUP BY
        department_id;
```

```
SELECT
        e.department_id,
        department_name,
        AVG(salary)
FROM
        employees e
INNER JOIN departments d ON d.department_id = e.department_id
GROUP BY
        e.department_id;
```

SQL COUNT function

- The SQL COUNT function is an aggregate function that returns the number of rows returned by a query. You can use the COUNT function in the SELECT statement to get the number of employees, the number of employees in each department, the number of employees who hold a specific job,

```
SELECT
    COUNT(*)
FROM
    employees;
```

```
SELECT
    COUNT(*)
FROM
    employees
WHERE
    department_id = 6;
```

```
SELECT
        department_id,
        COUNT(*)
FROM
        employees
GROUP BY
        department_id;
```

Excercise: Select departments that have more than five employees

```
SELECT
        e.department_id,
        department_name,
        COUNT(*)
FROM
        employees e
INNER JOIN departments d ON d.department_id = e.department_id
GROUP BY
        e.department_id
HAVING
        COUNT(*) > 5
ORDER BY
        COUNT(*) DESC;
```

Using distinct

```
SELECT
    COUNT(DISTINCT manager_id)
FROM
    employees;
```

SQL MAX function

- SQL provides the MAX function that allows you to find the maximum value in a set of values.

- use the MAX function to find the highest salary of employee in each department

```
SELECT
        department_id,
        MAX(salary)
FROM
        employees
GROUP BY
        department_id;
```

Exercise: the department that has employee whose highest salary is greater than 12000

```
SELECT
        d.department_id,
        department_name,
        MAX(salary)
FROM
        employees e
INNER JOIN departments d ON d.department_id = e.department_id
GROUP BY
        e.department_id
HAVING
        MAX(salary) > 12000;
```

department_id	department_name	MAX(salary)
2	Marketing	13000.00
8	Sales	14000.00
9	Executive	24000.00

SQL MIN function

- The SQL MIN function returns the minimum value in a set of values.
- To get the information of the employee who has the lowest salary

```
SELECT
    MIN(salary)
FROM
    employees;
```

```
SELECT
        employee_id,
        first_name,
        last_name,
        salary
FROM
        employees
WHERE
        salary = (
                SELECT
                        MIN(salary)
                FROM
                        employees
        );
```

Tech365

SQL SUM function

- The SQL SUM function is an aggregate function that returns the sum of all or distinct values.

```
SELECT
    SUM(salary)
FROM
    employees
WHERE
    department_id = 5;
```

```
SELECT
        e.department_id,
        department_name,
        SUM(salary)
FROM
        employees e
INNER JOIN departments d ON d.department_id = e.department_id
GROUP BY
        e.department_id
HAVING
        SUM(salary) > 30000
ORDER BY
        SUM(salary) DESC;
```

Tech365

Managing Database Objects

SQL Data types

- In a database, each column of a table has a specific data type. A data type specifies the type of data that column can hold such as character strings, numeric values, and date time values.

- SQL supplies a set of basic data types that you can use for defining columns of tables.

- The fixed-length **character** data type stores fixed-length character strings.

```
column_name CHARACTER(5)
```

- Varying-length character or **VARCHAR**

```
first_name VARCHAR(50)
```

Numeric Types

- Numeric values are stored in the columns with the type of numbers, typically referred to as NUMBER, INTEGER, REAL, and DECIMAL.

The following are the SQL numeric data types:

- BIT(n)
- BIT VARYING (n)
- DECIMAL (p,s)
- INTEGER
- SMALLINT
- BIGINT
- FLOAT(p,s)
- DOUBLE PRECISION (p,s)
- REAL(s)

Numeric Types

- **Decimal**
- The following defines the salary column with 12 digits which include 4 digits after the decimal point

```
salary DECIMAL (12,4)
```

- **Integer**
- The following defines the salary column with 12 digits which include 4 digits after the decimal point
- Integer data type stores whole numbers, both positive and negative. The examples of integers are 10, 0, -10, and 2010.

```
INT
```

Tech365
Leveraging technology for growth

Data Types contd...

Floating-point data types

- The floating-point data types represent approximate numeric values.
- The precision and scale of the floating point decimals are variable in lengths and virtually without limit.

```
FLOAT
FLOAT(10)
FLOAT(50)
```

Date and Time types

- The date and time data types are used to store information related to dates and times

The date value generally is specified in the form:

```
'YYYY-DD-MM'
```

For example, the following `DATE` value is `December 31, 2020` :

```
'2020-12-31'
```

Tech365

Creating tables

- A table is a collection of data stored in a database. A table consists of columns and rows.

```
CREATE TABLE table_name(
     column_name_1 data_type default value column_constraint,
     column_name_2 data_type default value column_constraint,
     ...,
     table_constraint
);
```

```
CREATE TABLE courses (
    course_id INT AUTO_INCREMENT PRIMARY KEY,
    course_name VARCHAR(50) NOT NULL
);
```

```
CREATE TABLE trainings (
    employee_id INT,
    course_id INT,
    taken_date DATE,
    PRIMARY KEY (employee_id , course_id)
);
```

Auto Increment

- **MySQL** uses AUTO_INCREMENT property to define an auto-increment column

```
CREATE TABLE leave_requests (
    request_id INT AUTO_INCREMENT,
    employee_id INT NOT NULL,
    start_date DATE NOT NULL,
    end_date DATE NOT NULL,
    leave_type INT NOT NULL,
    PRIMARY KEY(request_id)
);
```

In PostgreSQL or ORACLE

```
CREATE TABLE leave_requests (
    request_id INT GENERATED BY DEFAULT AS IDENTITY,
    employee_id INT NOT NULL,
    start_date DATE NOT NULL,
    end_date DATE NOT NULL,
    leave_type INT NOT NULL,
    PRIMARY KEY(request_id)
);
```

SQL INSERT statement

- QL provides the INSERT statement that allows you to insert one or more rows into a table.

```
INSERT INTO dependents (
        first_name,
        last_name,
        relationship,
        employee_id
)
VALUES
        (
                'Cameron',
                'Bell',
                'Child',
                192
        ),
        (
                'Michelle',
                'Bell',
                'Child',
                192
        );
```

SQL UPDATE statement

- To change existing data in a table, you use the UPDATE statement. The following shows the syntax of the UPDATE statement

```
UPDATE employees
SET
    last_name = 'Lopez'
WHERE
    employee_id = 192;
```

Tech365

SQL DELETE statement

- To remove one or more rows from a table, you use the DELETE statement.

```
DELETE
FROM
        employees
WHERE
        employee_id = 192;
```

```
DELETE FROM dependents
WHERE
    employee_id IN (100 , 101, 102);
```

SQL ALTER TABLE ADD column

To add one or more columns to a table, you need to perform the following steps:

First, specify the table that you want to add column denoted by the table_name after the ALTER TABLE clause.
Second, place the new column definition after the ADD clause. If you want to specify the order of the new column in the table, you can use the optional clause AFTER existing_column.
Note that if you omit the AFTER clause, all the new columns will be added after the last column of the table.

```
ALTER TABLE courses ADD credit_hours INT NOT NULL;
```

```
ALTER TABLE courses
ADD fee NUMERIC (10, 2) AFTER course_name,
ADD max_limit INT AFTER course_name;
```

Tech365

SQL ALTER TABLE

- **SQL ALTER TABLE MODIFY column**

```
ALTER TABLE courses
MODIFY fee NUMERIC (10, 2) NOT NULL;
```

- **SQL ALTER TABLE DROP columns**

```
ALTER TABLE courses
DROP COLUMN max_limit,
DROP COLUMN credit_hours;
```

Adding column and dropping table

- **SQL ADD COLUMN clause**

```
ALTER TABLE candidates
ADD COLUMN home_address VARCHAR(255),
ADD COLUMN dob DATE,
ADD COLUMN linkedin_account VARCHAR(255);
```

- **SQL DROP TABLE clause**

```
DROP TABLE emergency_contacts;
```

SQL TRUNCATE TABLE

- To delete all rows from a big table fast, you use the following TRUNCATE TABLE statement

```
TRUNCATE TABLE table_name;
```

SQL Constraints

SQL Primary Key

- A table consists of columns and rows. Typically, a table has a column or set of columns whose values uniquely identify each row in the table. This column or the set of columns is called the primary key.
- Each table has **one and only one primary key**. The primary key does not accept NULL or duplicate values.

```
CREATE TABLE projects (
    project_id INT PRIMARY KEY,
    project_name VARCHAR(255),
    start_date DATE NOT NULL,
    end_date DATE NOT NULL
);
```

```
CREATE TABLE projects (
    project_id INT,
    project_name VARCHAR(255),
    start_date DATE NOT NULL,
    end_date DATE NOT NULL,
    CONSTRAINT pk_id PRIMARY KEY (project_id)
);
```

Tech365

SQL Foreign Key

- A foreign key is a column or a group of columns that enforces a link between the data in two tables. In a foreign key reference, the primary key column (or columns) of the first table is referenced by the column (or columns) of the second table. The column (or columns) of the second table becomes the foreign key.

- You use the FOREIGN KEY constraint to create a foreign key when you create or alter table. Let's take a simple example to get a better understanding.

Foreign key example

```sql
CREATE TABLE projects (
    project_id INT AUTO_INCREMENT PRIMARY KEY,
    project_name VARCHAR(255),
    start_date DATE NOT NULL,
    end_date DATE NOT NULL
);
```

```sql
CREATE TABLE project_milestones (
    milestone_id INT AUTO_INCREMENT PRIMARY KEY,
    project_id INT,
    milestone_name VARCHAR(100),
    FOREIGN KEY (project_id)
        REFERENCES projects (project_id)
);
```

Adding or removing foreign key in existing table

- Adding foreign key

```
ALTER TABLE project_milestones
ADD CONSTRAINT fk_project FOREIGN KEY(project_id)
   REFERENCES projects(project_id);
```

- Removing foreign key

```
ALTER TABLE table_name
DROP FOREIGN KEY fk_name;
```

SQL UNIQUE Constraint

- Sometimes, you want to make sure that the values in a column or a set of columns are not duplicate. For example, duplicate emails in the employees table are not acceptable.

- Since the email column is not the part of the primary key, the only way to prevent duplicate values in the email column is to use a UNIQUE constraint.

- By definition, an SQL UNIQUE constraint defines a rule that prevents duplicate values stored in specific columns that do not participate a primary key.

UNIQUE vs. PRIMARY KEY constraints

You can have at most one PRIMARY KEY constraint whereas you can have multiple UNIQUE constraints in a table. In case you have multiple UNIQUE constraints in a table, all UNIQUE constraints must have a different set of columns.

Different from the PRIMARY KEY constraint, the UNIQUE constraint allows NULL values. It depends on the RDBMS to consider NULL values are unique or not.

	PRIMARY KEY constraint	**UNIQUE constraint**
The number of constraints	One	Many
NULL values	Do not allow	Allow

Adding unique constraint

```sql
CREATE TABLE users (
    user_id INT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(255) NOT NULL UNIQUE,
    password VARCHAR(255) NOT NULL
);
```

```sql
ALTER TABLE users
ADD email VARCHAR(255) UNIQUE;
```

Check constraint

- A CHECK constraint is an integrity constraint in SQL that allows you to specify that a value in a column or set of columns must satisfy a Boolean expression.

```
CREATE TABLE products (
        product_id INT PRIMARY KEY,
        product_name VARCHAR (255) NOT NULL,
        selling_price NUMERIC (10, 2) CHECK (selling_price > 0),
        cost NUMERIC (10, 2) CHECK (cost > 0),
        CONSTRAINT valid_selling_price  CHECK (selling_price > cost)
);
```

NOT NULL constraint

- The NOT NULL constraint is a column constraint that defines the rule which constrains a column to have non-NULL values only.

```
CREATE TABLE training (
    employee_id INT,
    course_id INT,
    taken_date DATE NOT NULL,
    PRIMARY KEY (employee_id , course_id)
);
```

Thank you