الفهرس

EER diagram

- ملحوظة اول حاجة بنعملها ما بنشتغل على database جديدة اننا بنشوف علاقات الجدول

File / Edit / data base ← ١- بندوس على Data base اللى موجود في شريط البار

٢ - بنختار reverse engineer

عشان نعرف العلاقات ماشية ازاى عشان لما نيجى نكتب ال Queries

---

aperaters

المشغلات في SQL هي أدوات تُستخدم لتنفيذ عمليات مختلفة على البيانات. إليك الأنواع الرئيسية:

١. المشغلات الحسابية (Arithmetic): لإجراء العمليات الحسابية، مثل + (جمع)، - (طرح)، * (ضرب)، و / (قسمة).

```
SELECT salary + bonus AS total_income FROM employees
```

٢. مشغلات المقارنة (Comparison): لمقارنة القيم، مثل = (يساوي)، <> (لا يساوي)، > و <

```
SELECT * FROM employees WHERE salary >= 5000
```

٣. المشغلات المنطقية (Logical) : لدمج الشروط، مثل AND (و)، OR (أو)، NOT (ليس).

```
SELECT * FROM employees WHERE salary > 5000 AND department = 'Sales
```

٤. مشغلات النطاق والقيم المحددة :
- BETWEEN ... AND : للتحقق إذا كانت القيمة ضمن نطاق.
- IN : للتحقق إذا كانت القيمة ضمن مجموعة.
- LIKE : للبحث عن نمط معين بالنص.

```
SELECT * FROM employees WHERE salary BETWEEN 3000 AND 7000
SELECT * FROM employees WHERE department IN ('Sales', 'Marketing'
SELECT * FROM employees WHERE name LIKE 'A%'
```

٥. مشغل القيم الفارغة (NULL) : للتحقق إذا كانت القيمة فارغة IS NULL أو ليست فارغة IS NOT NULL .

```
SELECT * FROM employees WHERE bonus IS NULL
```

هذه المشغلات تساعد في كتابة استعلامات SQL فعّالة لتصفية وتحليل البيانات.

Execution Order

الفرق بين ترتيب الكتابة فى SQL(Writing Order) و ترتيب التنفيذ في SQL (Execution Order)
إليك توضيح الفرق:

ترتيب الكتابة (Writing Order)
- هذا هو الترتيب الذي نكتب في SQL. عادةً ما يكون بالشكل التالي:

```
SELECT columns
FROM table
WHERE condition
GROUP BY column
HAVING condition
ORDER BY column
LIMIT num
```

ترتيب التنفيذ (Execution Order)
- هذا هو الترتيب الفعلي الذي تنفذه قاعدة البيانات للاستعلام، ويختلف عن ترتيب الكتابة. يقوم SQL بمعالجة أجزاء الاستعلام بهذا الترتيب:

١. **FROM : تحديد الجداول.**
٢. **JOIN : دمج الجداول إن وجدت.**
٣. **WHERE : تصفية البيانات حسب الشروط.**
٤. **GROUP BY : تجميع البيانات.**
٥. **HAVING : تصفية المجموعات.**
٦. **SELECT : اختيار الأعمدة.**
٧. **ORDER BY : ترتيب النتائج.**
٨. **LIMIT : تحديد عدد السجلات المعروضة.**

مثال عملي

```
SELECT department, AVG(salary) AS avg_salary
FROM employees
WHERE salary > 3000
GROUP BY department
HAVING avg_salary > 5000
ORDER BY avg_salary DESC
;LIMIT 5
```

يكون ترتيب التنفيذ كالتالي:

```
FROM employees
WHERE salary > 3000
GROUP BY department
HAVING avg_salary > 5000
SELECT department, AVG(salary) AS avg_salary
ORDER BY avg_salary DESC
LIMIT 5
```

Clean code

1- **Add news lines for each key words**

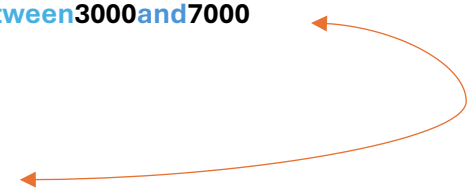
يعنى نضع كل من select ,from , where كل واحده فيهم في سطر لوحدو

2- **Uppercase all key words**

كل الحروف كابيتال

3- **Add space**

```
select*from employeeswhere salarybetween3000and7000
```



```
SELECT *
 FROM employees
WHERE  salary BETWEEN 3000 AND 7000
```

SQL Commands Guide

1. SELECT - FROM

Used to retrieve data from a database.
Syntax:

```
SELECT column1, column2
 FROM table_name ;
```

Example:

```
SELECT name, age
FROM employees;
```

```
SELECT * (All columns)
 FROM table_name
```

Example:

```
SELECT *
FROM employees;
```

2. WHERE

Used to filter records based on a condition.

Syntax

```
SELECT column1
FROM table_name
WHERE condition
```

Example:

```
SELECT name
 FROM employees
 WHERE age > 30;
```

3. COUNT, SUM, AVG, MIN, MAX

Aggregate functions used for calculations on groups of rows.

Example:

```
SELECT COUNT(*), AVG(salary), MAX(salary)
 FROM employees;
```

النتيجة : يعرض عدد الموظفين ومتوسط المرتب للموظفين واعلى مرتب ادفع

4.BETWEEN - NOT BETWEEN

وهو يساعد على تصفية البيانات بسهولة بين قيمتين محددتين، سواء كانت هذه القيم أرقامًا أو تواريخ

```
SELECT column_name(s)
FROM table_name
WHERE column_name BETWEEN value1 AND value2

SELECT *
 FROM employees
WHERE salary BETWEEN 3000 AND 7000
```

النتيجة: يعرض جميع الموظفين الذين تتراوح رواتبهم بين 3000 AND 7000

```
SELECT *
FROM orders
WHERE order_date BETWEEN '2023-01-01' AND '2023-12-31'
```

النتيجة: يعرض جميع الطلبات التي تم إجراؤها في عام ٢٠٢٣ .

NOT BETWEEN

- لاستبعاد قيم ضمن نطاق محدد.

```
SELECT *
 FROM employees
WHERE salary NOT BETWEEN 3000 AND 7000
```

النتيجة: يعرض الموظفين الذين لا تتراوح رواتبهم بين ٣٠٠٠ و ٧٠٠٠ .

ملخص

- BETWEEN : تشمل القيم في النطاق، وتُسهل تصفية البيانات بين قيمتين.
- NOT BETWEEN: تستثني القيم داخل النطاق.
- يمكن استخدام مع الأرقام، والتواريخ، والنصوص.

LIKE

`LIKE` هو عامل يستخدم للبحث عن نمط معين ضمن بيانات النصوص في الأعمدة، حيث يتيح لنا التحقق مما إذا كانت البيانات تحتوي على نمط معين من النصوص،

```
SELECT column_name(s)
FROM table_name
WHERE column_name LIKE pattern
```

`%` : يمثل أي عدد من الأحرف، ويمكن أن يكون صفرًا أو أكثر من الأحرف.
`_` : يمثل حرفًا واحدًا فقط.

```
SELECT *
FROM Customers
WHERE CustomerName LIKE 'A%'
```

سيظهر النتائج التي تبدأ بـ "A" مثل "Alice" و"Alex"

٢. البحث عن كلمة تنتهي بنمط معين نبحث هنا عن أسماء العملاء التي تنتهي بالحرف `n`:

```
SELECT *
FROM Customers
WHERE CustomerName LIKE '%n'
```

سيظهر النتائج التي تنتهي بحرف "n" مثل "Martin" و"John"

٣. البحث عن كلمة تحتوي على نمط معين في المنتصف للبحث عن أسماء العملاء التي تحتوي على الحروف `ar` في أي مكان:

```
SELECT *
FROM Customers
WHERE CustomerName LIKE '%ar%'
```

سيظهر النتائج التي تحتوي على "ar" مثل "Mary" و"Barbara".*

٤. البحث باستخدام `_` (حرف واحد) للبحث عن أسماء العملاء التي تتكون من ٤ أحرف وتبدأ بالحرف `J`:

```
SELECT *
FROM Customers
WHERE CustomerName LIKE 'J _ _ _'
```

سيظهر النتائج التي تبدأ بـ "J" وتتكون من ٤ أحرف مثل "John" و"Jack".*

٥. استخدام `LIKE` مع الحروف المتسلسلة للبحث عن أسماء العملاء التي تبدأ بحرف يتراوح بين `A` و`C`:

```
SELECT *
FROM Customers
WHERE CustomerName LIKE '[A-C]%'
```

هذا الشرط يعرض الأسماء التي تبدأ بـ "A" أو "B" أو "C" مثل "Alice" و"Barbara" و"Carl".*

AND/ OR/ IN

`AND`، `OR`، `IN` هي شروط تُستخدم في SQL لتحديد منطق فلترة البيانات ضمن أوامر `SELECT`، مما يتيح لنا استخراج البيانات بناءً على معايير متعددة. إليك كيفية استخدام كل منها مع أمثلة توضيحية.

`AND`

يستخدم الشرط `AND` لربط شرطين أو أكثر، بحيث يُظهر فقط الصفوف التي تحقق جميع الشروط المتاحة.

مثال
نريد اختيار العملاء الذين يعيشون في "القاهرة" وعمرهم أكبر من ٢٥ عامًا:

```
SELECT *
FROM Customers
WHERE City = 'Cairo' AND Age > 25 ;
```

النتيجة ستظهر فقط العملاء الذين يعيشون في "القاهرة" وأعمارهم فوق ٢٥.*

`OR`

يستخدم الشرط `OR` لربط شرطين أو أكثر، بحيث يُظهر الصفوف التي تحقق أيً شرط من الشروط.

مثال
نريد اختيار العملاء الذين يعيشون في "القاهرة" أو أعمارهم أكبر من ٢٥ عامًا:

```
SELECT *
FROM Customers
WHERE City = 'Cairo' OR Age > 25
```

النتيجة ستظهر جميع العملاء الذين يعيشون في "القاهرة" بغض النظر عن العمر، وأيضًا العملاء الذين أعمارهم أكبر من ٢٥ بغض النظر عن المدينة.*

`IN`

يُستخدم الشرط `IN` لتحديد عدة قيم للبحث عنها في عمود معين، وهو اختصار جيد لجملة `OR` عندما يكون لدينا العديد من القيم المحتملة.

مثال
نريد اختيار العملاء الذين يعيشون في "القاهرة" أو "الإسكندرية" أو "الجيزة":

```
SELECT *
FROM Customers
WHERE City IN ('Cairo', 'Alexandria', 'Giza')
```

النتيجة ستظهر جميع العملاء الذين يعيشون في أي من هذه المدن.

الجمع بين `AND` و`OR`

يمكنك دمج `AND` و`OR` معًا لإنشاء شروط أكثر تعقيدًا. استخدم الأقواس لتوضيح ترتيب تنفيذ الشروط.

مثال
نريد اختيار العملاء الذين يعيشون في "القاهرة" وأعمارهم أكبر من ٢٥، أو العملاء الذين يعيشون في "الإسكندرية":

```
SELECT *
FROM Customers
'WHERE (City = 'Cairo' AND Age > 25) OR City = 'Alexandria
```

النتيجة ستظهر العملاء الذين يعيشون في "القاهرة" وأعمارهم فوق ٢٥، وكذلك جميع العملاء الذين يعيشون في "الإسكندرية" بغض النظر عن العمر.

ملخص
- `AND`: يُظهر الصفوف التي تحقق جميع الشروط.
- OR: يُظهر الصفوف التي تحقق أي من الشروط.
- IN: يُحدد مجموعة من القيم المحتملة للبحث في عمود معين.

AS

`AS` في SQL هو عامل يستخدم لتعيين اسم مستعار (Alias) للأعمدة أو الجداول. هذا يجعل عرض البيانات أكثر وضوحًا ويجعل أسماء الأعمدة والجداول أسهل للفهم، خاصة عند استخدام عبارات معقدة أو عند جمع بيانات من جداول متعددة.

```
SELECT column_name AS alias_name
FROM table_name
```

مثال
لنفترض أن لدينا جدولًا باسم `Employees` يحتوي على الأعمدة `FirstName` و`LastName` و`Salary`. ونريد عرض الأسماء بشكل كامل في عمود يسمى `FullName` بدلًا من عرض `FirstName` و`LastName` كل على حدة:

```
SELECT FirstName || ' ' || LastName AS FullName, Salary
;FROM Employees
```

مثال آخر
نريد عرض عمود `Salary` كـ`EmployeeSalary` لتوضيح أن هذا العمود يمثل رواتب الموظفين:

```
SELECT FirstName, LastName, Salary AS EmployeeSalary
;FROM Employees
```

النتيجة ستعرض العمود `Salary` باسم `EmployeeSalary`.

استخدام `AS` مع الجداول

يمكن أيضًا استخدام `AS` لتعيين اسم مستعار للجداول، وهو مفيد عند كتابة استعلامات تحتوي على عدة جداول.

مثال

لنفترض أن لدينا جدولين، `Orders` و`Customers`، ونريد الربط بينهما. يمكننا استخدام أسماء مستعارة لتسهيل كتابة الاستعلام.

```
SELECT C.CustomerName, O.OrderID
FROM Customers AS C
JOIN Orders AS O
;ON C.CustomerID = O.CustomerID
```

في هذا المثال، `C` هو اسم مستعار لجدول `Customers` و`O` هو اسم مستعار لجدول `Orders`، مما يسهل الإشارة إلى الأعمدة أثناء كتابة الاستعلام.

الاستخدامات الرئيسية لـ `AS`

- **تبسيط عرض البيانات: يمكن استخدام `AS` لجعل أسماء الأعمدة أكثر وضوحًا للقارئ.**
- **التعامل مع الاستعلامات المعقدة: يسهّل `AS` العمل مع جداول وأعمدة متعددة، خاصة عند استخدام عبارات `JOIN`.**
- **تحسين المخرجات في التقارير: الأسماء المستعارة تجعل البيانات أوضح عند عرضها في تقارير أو شاشات عرض.**

ملخص

يُستخدم `AS` في SQL لتعيين أسماء مستعارة للأعمدة والجداول، مما يسهل قراءة الاستعلامات وفهمها، ويجعل نتائج البيانات أكثر وضوحًا.

. JOIN

Combines rows from two or more tables based on a related column.

Syntax:

```
SELECT columns
FROM table1
INNER JOIN table2 ON condition
```

Example:

```
SELECT employees.name, departments.department_name
FROM employees
 INNER JOIN
departments ON employees.department_id = departments.id;
```

. GROUP BY

Groups rows that have the same values into summary rows.

Syntax:

```
SELECT column, COUNT(*)
FROM table_name
GROUP BY column;
```

Example:

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

. HAVING

Filters records after GROUP BY.

Syntax:

```
SELECT column, COUNT(*)
FROM table
 GROUP BY column
 HAVING COUNT(*) > value
```

Example:

```
SELECT department_id, COUNT(*)
FROM employees
GROUP BY department_id
HAVING COUNT(*) > 5;
```

. ORDER BY

Sorts the result-set in ascending or descending order.

```
SELECT columns
FROM table_name
ORDER BY column1    ASC/DESC
```

ASC من الأصغر الى الأكبر / DESC من الأكبر الى الاصغر

Example:

```
SELECT name, age , salary
 FROM employees
 ORDER BY age DESC;
```

```
SELECT name, age , salary
 FROM employees
 ORDER BY age DESC, salary DESC ;
```

لو الأعمار اتشبهت هنرتب بالمرتبات بس في تشابة الأسعار بس

LIMIT / OFFSET

`LIMIT` و`OFFSET` في SQL تُستخدمان للتحكم في عدد الصفوف التي يتم عرضها من نتيجة الاستعلام، وتحديد الصفوف التي يبدأ العرض منها. هذه الأوامر مفيدة خاصة عند التعامل مع كميات كبيرة من البيانات، أو عند تقسيم النتائج إلى صفحات.

الصيغة العامة لـ `LIMIT` و`OFFSET`

```
SELECT column_name(s)
FROM table_name
;LIMIT number_of_rows OFFSET offset_value
```

- `LIMIT` : يحدد عدد الصفوف التي سيتم عرضها في النتائج.
- `OFFSET` : يحدد عدد الصفوف التي سيتم تخطيها قبل بدء عرض النتائج.

> ملاحظة : بعض قواعد بيانات SQL، مثل MySQL وPostgreSQL، تدعم الصيغة التالية المختصرة بدون `OFFSET`:

```
SELECT column_name(s)
FROM table_name
;LIMIT offset_value, number_of_rows <
<
```

أمثلة على استخدام `LIMIT` و`OFFSET`

مثال
نريد جلب أول ٥ صفوف من جدول `Employees`:

```
SELECT *
FROM Employees
;LIMIT 5
```

ستظهر النتيجة فقط أول ٥ صفوف من الجدول.

استخدام `LIMIT` مع `OFFSET`
يتيح `OFFSET` تخطي عدد معين من الصفوف ثم عرض عدد محدد بعدها. يكون هذا مفيدًا جدًا للتنقل بين صفحات البيانات.

مثال
نريد جلب ٥ صفوف، ولكن مع تخطي أول ١٠ صفوف في جدول `Employees`:

```
SELECT *
FROM Employees
;LIMIT 5 OFFSET 10 = LIMIT 10,5
```

في هذا المثال، سيتم تخطي أول ١٠ صفوف وعرض ٥ صفوف بعدها فقط.

ملخص

- **`LIMIT` : يحدد عدد الصفوف المطلوب عرضها.**
- **`OFFSET` : يحدد عدد الصفوف التي سيتم تخطيها قبل عرض النتائج.**
- **استخدامهما معًا : يساعد في تقسيم البيانات لعرضها بشكل تدريجي أو على صفحات، وهو مفيد في بناء واجهات التطبيقات والمواقع التي تتطلب عرض كميات كبيرة من البيانات.**

SUBQUERY

هو بديل JOIN بنستخدموا لما نكون عاوز معلومة من جدول اخر
بنكتبها في WHERE

A query within another query.
Syntax
:

```
SELECT column
FROM table
WHERE column = (SELECT column FROM table WHERE condition);
```

Example:

```
SELECT *
 FROM products
WHERE categoryid = (SELECT categoryid FROM categoryid WHERE
categoryname='Beverages');
```

```
SELECT *
 FROM products AS P INNERJOIN category AS C
ON  P. categoryid=C. categoryid
WHERE C. categoryname='Beverages' ;
```

Window Functions

الدوال النافذة (Window Functions) في SQL هي دوال تُستخدم لإجراء عمليات حسابية على مجموعة من الصفوف المرتبطة بكل صف في الجدول، دون الحاجة إلى تجميع الصفوف باستخدام `GROUP BY`. تُعَدُّ هذه الدوال مفيدة في تحليل البيانات وحسابات مثل المجموع التراكمي، المتوسط المتحرك، وترتيب الصفوف. يتم استخدام `OVER` لتحديد نطاق (أو نافذة) الصفوف التي سيتم تطبيق العملية عليها.

```
SELECT column_name, window_function() OVER (PARTITION BY
column_name ORDER BY column_name) AS alias_name
;FROM table_name
```

أشهر الدوال النافذة

- `ROW_NUMBER()` : يُعطي رقمًا تسلسليًا لكل صف في كل قسم.
- `RANK()` : يعطي ترتيبًا للصفوف في كل قسم، مع السماح بتكرار الترتيب.
- `DENSE_RANK()` : يشبه `RANK()` لكنه لا يتخطى أرقام الترتيب.
- `SUM()` و`AVG()` : لحساب المجاميع أو المتوسطات على نافذة من الصفوف.
- `LAG()` و`LEAD()` : للوصول إلى قيمة الصف السابق أو اللاحق.

`()ROW_NUMBER` .

نستخدم `()ROW_NUMBER` لترقيم الصفوف ضمن كل مجموعة حسب ترتيب معين.

مثال
نفترض أن لدينا جدولًا باسم `Employees` يحتوي على الأعمدة `Department` و`Salary`.
نريد إعطاء رقم لكل موظف في كل قسم حسب ترتيب الرواتب.

```
,SELECT Department, EmployeeName, Salary
ROW_NUMBER() OVER (PARTITION BY Department ORDER BY Salary
DESC) AS RowNumber
;FROM Employees
```

في هذا المثال، يتم تقسيم الصفوف حسب `Department`، وترتيب الموظفين في كل قسم حسب `Salary` بترتيب تنازلي، ثم إعطاء رقم تسلسلي.

`()RANK` .

يستخدم `()RANK` لترتيب الصفوف داخل كل مجموعة، لكنه يسمح بتكرار الترتيب.

مثال
نريد ترتيب الموظفين في كل قسم حسب الرواتب مع السماح بتكرار الرتب عند تساوي الرواتب.

```
,SELECT Department, EmployeeName, Salary
RANK() OVER (PARTITION BY Department ORDER BY Salary DESC) AS
Rank
;FROM Employees
```

في هذه النتيجة، سيحصل الموظفون الذين يتقاضون نفس الراتب على نفس الترتيب، وستتخطى الرتب عند تكرارها.

`()SUM`

مع نافذه يمكن استخدام `()SUM` لحساب المجموع التراكمي للرواتب.

مثال
نريد حساب مجموع الرواتب التراكمي للموظفين حسب الراتب.

```
SELECT EmployeeName, Salary,
SUM(Salary) OVER (ORDER BY Salary) AS CumulativeSalary
FROM Employees;
```

سيعرض هذا مجموعًا تراكميًا لرواتب الموظفين حسب الترتيب، مما يظهر إجمالي الرواتب بشكل تصاعدي.

`()LAG` و `()LEAD` .

يمكن استخدام `()LAG` للحصول على قيمة من الصف السابق، و`()LEAD` للحصول على قيمة من الصف اللاحق.

مثال
نريد معرفة راتب الموظف السابق والموظف اللاحق لكل موظف حسب ترتيب الرواتب.

```
SELECT EmployeeName, Salary,
LAG(Salary) OVER (ORDER BY Salary) AS PreviousSalary,
LEAD(Salary) OVER (ORDER BY Salary) AS NextSalary
FROM Employees;
```

في هذه النتيجة، سيتم عرض راتب الموظف السابق وراتب الموظف اللاحق بجانب كل راتب.

ملخص
الدوال النافذة في SQL توفر إمكانيات قوية للتحليل المتقدم دون الحاجة إلى تجميع الصفوف. وتسمح بإجراء حسابات على صفوف متعددة لكل صف، مما يجعلها مثالية للتقارير والإحصاءات المتقدمة.

INSERT

في SQL، يُستخدم الأمر `INSERT` لإضافة بيانات جديدة إلى جداول قاعدة البيانات. يتيح لك هذا الأمر إدخال صفوف جديدة أو مجموعة من الصفوف في جدول معين، ويُستخدم بالشكل التالي:

الصيغة الأساسية لـ `INSERT`:

```
INSERT INTO table_name (column1, column2, column3, ...)
;VALUES (value1, value2, value3, ...)
```

- table_name : اسم الجدول الذي ترغب في إدخال البيانات فيه.
- column1, column2, ... : أسماء الأعمدة التي سيتم إدخال القيم فيها. (يمكنك اختيار عدم تحديد الأعمدة في حالة إدخال قيم لجميع الأعمدة في الجدول بترتيبها)
- value1, value2, ... : القيم التي ستُدخل لكل عمود في الصف.

مثال:
افترض أن لدينا جدولًا باسم `employees` يحتوي على الأعمدة التالية: `id`, `name`, `age`, و `department`.

إدخال صف جديد بذكر أسماء الأعمدة:

```
INSERT INTO employees (id, name, age, department)
;VALUES (1, 'Ali', 30, 'Sales')
```

في هذا المثال، نُدخل صفًا جديدًا في جدول `employees`، حيث:

- `id` هو ١.
- `name` هو "Ali".
- `age` هو ٣٠.
- `department` هو "Sales".

إدخال صف جديد بدون ذكر أسماء الأعمدة:

```
INSERT INTO employees
;VALUES (2, 'Mona', 25, 'Marketing')
```

في هذه الحالة، نتوقع أن القيم مرتبة تمامًا حسب ترتيب الأعمدة في الجدول. لذا، القيم المدخلة ستذهب مباشرة إلى الأعمدة بالترتيب الموجود في تعريف الجدول.

إدخال عدة صفوف في عملية واحدة:

```
INSERT INTO employees (id, name, age, department)
VALUES
,('Ahmed', 28, 'IT' ,٣)
;('Sara', 35, 'Finance' ,٤)
```

يتيح لنا هذا المثال إدخال عدة صفوف دفعة واحدة.

ملاحظات إضافية:
- إذا كان أحد الأعمدة يحتوي على خاصية `AUTO_INCREMENT` مثل عمود `id`، يمكنك تجنب إدخال قيمة له، وسيقوم النظام تلقائيًا بإعطائه قيمة جديدة.

مثال على `AUTO_INCREMENT`:

```
INSERT INTO employees (name, age, department)
;VALUES ('Fadi', 29, 'HR')
```

في هذا المثال، إذا كان `id` هو `AUTO_INCREMENT`، سيتم تعيين قيمة فريدة تلقائيًا له، ولن تحتاج لإدخاله.

إدخال بيانات باستخدام نتائج استعلام آخر:
في بعض الأحيان، قد تحتاج إلى إدخال بيانات من جدول آخر أو نتيجة استعلام.

```
INSERT INTO new_employees (name, age, department)
SELECT name, age, department FROM employees WHERE department =
;"Sales
```

في هذا المثال، يتم نقل بيانات جميع الموظفين في قسم "Sales" من جدول `employees` إلى جدول `new_employees`.

Update/set

في SQL، يُستخدم الأمر `UPDATE` لتحديث البيانات الموجودة في الجداول. يتيح لك هذا الأمر تعديل قيم معينة في الأعمدة لصف أو أكثر بناءً على شروط محددة.

الصيغة الأساسية لأمر `UPDATE`:

```
UPDATE table_name
,SET column1 = value1, column2 = value2
;WHERE condition
```

- table_name : اسم الجدول الذي ترغب في تحديث بياناته.
- SET : تُستخدم لتحديد الأعمدة التي ستتغير قيمها والقيم الجديدة لها.
- column1 = value1 : تعني أن العمود `column1` سيتم تعيينه إلى `value1`.
- WHERE : تُستخدم لتحديد الشروط التي يجب أن تتطابق مع الصفوف التي سيتم تحديثها. إذا لم تُستخدم، سيتم تحديث كل الصفوف في الجدول!

مثال:
افترض أن لدينا جدولًا باسم `employees` يحتوي على الأعمدة: `id`, `name`, `age`, و`department`.

تحديث قيمة واحدة لصف معين:

```
UPDATE employees
SET age = 31
;WHERE id = 1
```

في هذا المثال، نُحدث عمر الموظف الذي يحمل `id` = `1` ليصبح `٣١`.

تحديث عدة أعمدة في صف معين:

```
UPDATE employees
'SET age = 32, department = 'Marketing
;WHERE id = 2
```

في هذا المثال، سيتم تحديث صف الموظف الذي يحمل `id` = 2` بحيث:
- يصبح `age` = 32`.
- يصبح `department` = 'Marketing'

تحديث عدة صفوف:
يمكنك أيضًا تحديث عدة صفوف باستخدام شروط عامة في جملة `WHERE`.

```
UPDATE employees
'SET department = 'Sales
;'WHERE department = 'Marketing
```

في هذا المثال، سيتم تغيير قسم جميع الموظفين الذين ينتمون إلى "Marketing" ليصبحوا في قسم "Sales".

ملاحظة:
- عدم استخدام `WHERE` : إذا قمت بتجاهل شرط `WHERE`، سيتم تحديث جميع الصفوف في الجدول، وهذا قد يؤدي إلى تغيير البيانات بشكل غير مرغوب فيه.

مثال بدون `WHERE`:

```
UPDATE employees
;'SET department = 'General
```

في هذا المثال، سيتم تعيين القسم "General" لكل الموظفين في الجدول `employees`.

استخدام `UPDATE` مع `SET` يعطيك مرونة كبيرة في تعديل البيانات، ولكن يجب الانتباه جيدًا للشروط لتجنب التحديثات غير المقصودة.

DROP/DELETE

تُستخدم أوامر `DELETE` و`DROP` لإزالة البيانات، لكنهما يختلفان في النطاق والوظيفة. دعنا نستعرض كل منهما بالتفصيل.

١. `DELETE`

- الوصف : يُستخدم الأمر `DELETE` لإزالة صفوف معينة من جدول. يمكنك تحديد صفوف معينة باستخدام جملة `WHERE`. إذا لم تُستخدم `WHERE`، سيتم حذف جميع الصفوف في الجدول، ولكن هيكل الجدول نفسه سيظل موجودًا.
- الاستخدام : يُستخدم لحذف بيانات محددة من جدول دون التأثير على هيكل الجدول.

مثال:

```
DELETE FROM employees
WHERE id = 1;
```

في هذا المثال:
- سيؤدي هذا الأمر إلى حذف الصف الذي يحمل `id` = `1` من جدول `employees`.

حذف جميع الصفوف في الجدول:

```
DELETE FROM employees;
```

في هذا المثال:
- سيتم حذف جميع الصفوف في جدول `employees`، ولكن سيظل هيكل الجدول نفسه موجودًا.

٢. `DROP`

- الوصف : يُستخدم الأمر `DROP` لإزالة كائنات قاعدة البيانات بالكامل، مثل الجداول، قواعد البيانات، أو الفهارس. عند استخدام `DROP` على جدول، يتم حذف الجدول بالكامل مع جميع البيانات والمعلومات المتعلقة به، ولا يمكن استعادته بسهولة.
- الاستخدام : يُستخدم عندما ترغب في إزالة جدول أو قاعدة بيانات بالكامل.

مثال:

```
DROP TABLE employees;
```

في هذا المثال:
- سيؤدي هذا الأمر إلى حذف جدول `employees` بالكامل مع جميع البيانات الموجودة فيه.

حذف قاعدة بيانات كاملة:

```
DROP DATABASE company_db;
```

في هذا المثال:
- سيؤدي هذا الأمر إلى حذف قاعدة البيانات `company_db` وكل الجداول والبيانات الموجودة بداخلها.

الاختلافات الرئيسية بين `DELETE` و`DROP`:

١. النطاق :
- `DELETE`: يحذف صفوف من جدول مع الإبقاء على هيكل الجدول.
- `DROP`: يحذف كائنات قاعدة البيانات بالكامل، بما في ذلك البيانات والبنية.

٢. الاسترجاع :
- `DELETE`: يمكن استخدام المعاملات (مثل `COMMIT` و`ROLLBACK`) لإمكانية استرجاع البيانات المحذوفة إذا تم تنفيذها في سياق معاملة.
- `DROP`: لا يمكن استرجاع الكائنات المحذوفة بسهولة، وهي دائمة.

٣. الأداء :
- `DELETE` يمكن أن يكون أبطأ إذا كان هناك عدد كبير من الصفوف المراد حذفها، حيث يقوم بتسجيل كل صف تم حذفه.
- `DROP` يعمل عادةً بشكل أسرع لأنه يحذف الكائن بالكامل دون الحاجة لتسجيل التغييرات.

ملاحظات:
- قبل تنفيذ أوامر `DELETE` و`DROP`، يجب التأكد من أن لديك النسخ الاحتياطية اللازمة، خصوصًا عند استخدام `DROP`، لأنه لا يمكن التراجع عن هذا الأمر بسهولة.
- من الجيد دائمًا مراجعة التعليمات الخاصة بقاعدة البيانات الخاصة بك، لأن بعض أنظمة إدارة قواعد البيانات قد يكون لها سلوكيات أو قيود مختلفة بخصوص هذه الأوامر.