Data Manipulation Language (DML):

تشمل تعليمات DML:

1. إضافة أسطر جديدة إلى جدول.
2. تعديل أسطر موجودة في جدول.
3. حذف أسطر موجودة في جدول.

1- إضافة سطر جديد إلى جدول:

يتم إضافة سطر جديد إلى الجدول باستخدام عبارة INSERT، إن الشكل العام لهذه العبارة هو:

```
INSERT INTO   table [(column [, column...])]
VALUES        (value [, value...]);
```

مثال:

```
INSERT INTO departments(department_id, department_name,
                        manager_id, location_id)
VALUES      (70, 'Public Relations', 100, 1700);
1 row created.
```

نلاحظ من المثال السابق أنه تم إضافة سطر وحيد إلى الجدول departments وتحديداً إلى الأعمدة department_id , department_name, manager_id , location_id من هذا الجدول وذلك بالقيم الموضحة بالمثال لكل عمود بالترتيب، كما نلاحظ أن القيم النصية وقيم التاريخ لا بد من وضعها ضمن علامات تنصيص.

مثال:

```
INSERT INTO  departments (department_id,
                          department_name    )
VALUES       (30, 'Purchasing');
1 row created.
```

نلاحظ في المثال السابق أنه تم إضافة سطر واحد إلى جدول departments وإلى العمودين department_id , department_name فقط من هذا الجدول بالقيمتين الموضحتين بالمثال.

لكن ماذا عن بقية أعمدة الجدول، بما أنه لم يتم إلحاق أي قيم بتلك الأعمدة فإن القيمة الجديدة ستكون NULL، وهنا يجب الانتباه إلى امكانية قبول هذا العمود لقيم NULL، أي لا يوجد شرط NOT

NULL مفروض على ذلك العمود من الجدول، ويمكن التحقق من ذلك عن طريق استخدام عبارة DESCRIBE لهذا الجدول.

ويمكن إضافة ذلك السطر بطريقة أخرى صريحة وهي:

```
INSERT INTO  departments
VALUES       (100, 'Finance', NULL, NULL);
1 row created.
```

نلاحظ بما أنه لم يتم ذكر أعمدة بعد اسم الجدول، فإن إلحاق القيم سيكون بالترتيب الافتراضي لأعمدة الجدول وكما يظهر عند استخدام الأمر DESCRIBE لهذا الجدول.

ملاحظات يجب أخذها بعين الاعتبار أثناء إضافة قيم جديدة إلى جدول:

1. لا بد من إلحاق قيم جديدة بالأعمدة المفروض عليها شرط NOT NULL وعدم تركها فارغة.
2. عدم إلحاق قيم مكررة بأعمدة هي مفتاح أساسي PRIMARY KEY، أو مفروض عليها شرط UNIQUE (عدم السماح بتكرار القيم).
3. عدم تجاوز شرط FOREIGN KEY أي لايمكن إلحاق قيم جديدة لأعمدة في الجدول الابن قبل إضافتها إلى الجدول الأب.
4. عدم تجاوز شرط CHECK وهو شرط التحقق من الصحة المفروض على العمود، أثناء إنشاء الجدول.
5. عدم إلحاق قيم ذات أنواع بيانات يختلف عن نوع بيانات العمود.

استخدام القيم الخاصة:

يمكن استخدام التوابع لإلحاق قيم جديدة بأعمدة الجدول

مثال:

```
INSERT INTO employees (employee_id,
                       first_name, last_name,
                       email, phone_number,
                       hire_date, job_id, salary,
                       commission_pct, manager_id,
                       department_id)
VALUES                 (113,
                       'Louis', 'Popp',
                       'LPOPP', '515.124.4567',
                       SYSDATE, 'AC_ACCOUNT', 6900,
                       NULL, 205, 100);
1 row created.
```

مثال 2:

```
INSERT INTO employees
VALUES       (114,
              'Den', 'Raphealy',
              'DRAPHEAL', '515.127.4561',
              TO_DATE('FEB 3, 1999', 'MON DD, YYYY'),
              'AC_ACCOUNT', 11000, NULL, 100, 30);
1 row created.
```

نلاحظ من الأمثلة السابقة أن القيم الجديدة المراد إضافتها إلى الجدول دائماً ثابتة، لكن ماذا لو أردنا أن تكون متغيرة حسب إرادة المستخدم عندها لا بد من استخدام الرمز & يليه مباشرة اسم متحول يعبر عن القيمة التي سيُطلب إدخالها من المستخدم.

مثال:

```
INSERT INTO departments
            (department_id, department_name, location_id)
VALUES      (&department_id, '&department_name',&location);
```

نلاحظ أن القيم النصية تمت إحاطتها بعلامات تنصيص

عند تنفيذ المثال السابق ستظهر رسالة أولى تطالب المستخدم بإدخال قيمة تمثل رقم القسم، ثم تظهر رسالة أخرى تطالب بإدخال اسم القسم....

إضافة أسطر جديدة إلى جدول من جدول آخر:

وهنا يتم استبدال عبارة VALUES باستعلام فرعي، لكن لا بد من يتطابق عدد الأعمدة المستخدمة بعد اسم الجدول ونوع بياناتها مع تلك المستخدمة في الاستعلام الفرعي.

مثال:

```
INSERT INTO sales_reps(id, name, salary, commission_pct)
  SELECT employee_id, last_name, salary, commission_pct
  FROM   employees
  WHERE  job_id LIKE '%REP%';
```

```
INSERT INTO copy_emp
   SELECT *
   FROM employees;
```

ويمكن استخدام الاستعلام الفرعي SUBQUERY في تعليمة INSERT بالشكل:

مثال:

```
INSERT INTO
        (SELECT employee_id, last_name,
                email, hire_date, job_id, salary,
                department_id
         FROM   employees
         WHERE  department_id = 50)
VALUES (99999, 'Taylor', 'DTAYLOR',
        TO_DATE('07-JUN-99', 'DD-MON-RR'),
        'ST_CLERK', 5000, 50);
```

وهنا يجب أن يتطابق عدد الأعمدة في الاستعلام الفرعي مع عدد القيم المستخدمة في عبارة VALUES.

2- **تعديل بيانات أسطر موجودة:**

ويتم ذلك عن طريق استخدام العبارة UPDATE، والتي لها الشكل العام التالي:

```
UPDATE      table
SET         column = value [, column = value, ...]
[WHERE      condition];
```

مثال:

```
UPDATE employees
SET    department_id = 70
WHERE  employee_id = 113;
```

لاحظ أنه إذا لم يتم ذكر عبارة WHERE فإن كامل أسطر الجدول سيتم تعديلها.

```
UPDATE   copy_emp
SET      department_id = 110;
```

إذا قمنا بعرض بيانات الجدول COPY_EMP سنلاحظ أن جميع أسطر الجدول قد تم تعديلها.

تعديل بيانات صفوف بالاعتماد على بيانات جدول آخر:

```
UPDATE   copy_emp
SET      department_id  =  (SELECT department_id
                           FROM employees
                           WHERE employee_id = 100)
WHERE    job_id         =  (SELECT job_id
                           FROM employees
                           WHERE employee_id = 200);
```

عند تعديل بيانات في جدول يجب الأخذ بعين الاعتبار الملاحظات التي تم ذكرها سابقاً عند الحاق بيانات جديدة بجدول.

مثال: حاول تنفيذ المثال التالي ولاحظ رسالة الخطأ التي ستظهر

```
UPDATE  employees
SET     department_id  = 55
WHERE   department_id  = 110;
```

السبب في الخطأ هو محاولة استخدام رقم قسم جديد (55) وهو غير موجود في الجدول الأب DEPARTMENTS.

3- **حذف أسطر من جدول:**

تستخدم عبارة DELETE لحذف الصفوف من الجدول، ولها الشكل العام التالي:

```
DELETE [FROM]   table
[WHERE          condition];
```

عند حذف بيانات من جدول يجب الأخذ بعين الاعتبار الملاحظات التي تم ذكرها سابقاً عند الحاق بيانات جديدة بجدول.

مثال:

```
DELETE FROM departments
WHERE  department_name = 'Finance';
```

في حال عدم تحديد شرط للحذف (..... WHERE)، فإنه سيتم حذف كامل أسطر الجدول.

مثال:

```
DELETE FROM  copy_emp;
22 rows deleted.
```

حذف بيانات من جدول اعتماداً على جدول آخر:

```
DELETE FROM employees
WHERE  department_id =
                       (SELECT department_id
                        FROM   departments
                        WHERE  department_name LIKE '%Public%');
```

استخدام القيمة الافتراضية للعمود:

يمكن استخدام القيمة الافتراضية للعمود التي تم تعيينها عند إنشاء الجدول، عن طريق استخدام الكلمة المحجوزة DEFAULT ، ويمكن استخدام هذه الميزة في كل من عبارتي INSERT , UPDATE.

ملاحظة في حال لم يكن هناك قيمة افتراضية للعمود وتم استخدام الكلمة DEFAULT عندها يتم تعيين القيمة NULL كقيمة جديدة لهذا الحقل من الجدول.

مثال:

```
INSERT INTO departments
  (department_id, department_name, manager_id)
VALUES (300, 'Engineering', DEFAULT);
```

```
UPDATE departments
SET manager_id = DEFAULT WHERE department_id = 10;
```

Database Transactions:

إن أي من التغييرات السابقة التي يتم إجراؤها على قاعدة البيانات من إضافة أوتعديل أوحذف تدعى **Transaction** ، وهذه التغيرات بحاجة إلى تثبيت **COMMIT** أو إلغاء **ROLLBACK**

Statement	Description
COMMIT	Ends the current transaction by making all pending data changes permanent
SAVEPOINT *name*	Marks a savepoint within the current transaction
ROLLBACK	ROLLBACK ends the current transaction by discarding all pending data changes
ROLLBACK TO *SAVEPOINT name*	ROLLBACK TO SAVEPOINT rolls back the current transaction to the specified savepoint, thereby discarding any changes and or savepoints created after the savepoint to which you are rolling back. If you omit the TO SAVEPOINT clause, the ROLLBACK statement rolls back the entire transaction. As savepoints are logical, there is no way to list the savepoints you have created.

الاستعلامات (الاستفسارات – Quries)

تعليمة SELECT: تستخدم للاستفسار عن البيانات ، ويمكن أن تتضمن:

الرمز	المعنى
*	اختيار جميع الأعمدة
DISTINCT	عدم تكرار نفس السطر في النتيجة
column | expression	لاختيار العمود المسمى أو التعبير المطبق .
Alias	الاسم المستعار .

مثال (1) : اختيار جميع الأعمدة...

```
SELECT *
FROM   departments;
```

مثال (2) : اختيار أعمدة محددة.

```
SELECT department_id, location_id
FROM   departments;
```

كتابة تعليمات SQL:

- ✓ تعليمات SQL ليست حساسة لحالة الأحرف.
- ✓ يمكن أن تُكتب كامل عبارة SQL على سطر واحد ، أو تكتب على عدة أسطر.
- ✓ لا يمكن اختصار الكلمات المحجوزة ، أو تجزئتها في عدة أسطر.
- ✓ توضع الجمل عادة في أسطر منفصلة (مثل SELECT في سطر ، FROM في سطر آخر ، وهكذا ..).

التعابير الحسابية: يمكن استخدام العمليات الحسابية (+، – ، *، /) على الاعمدة في عبارة select،

- ✓ عمليتي الضرب والقسمة تملكان أفضلية أعلى من عمليتي الجمع والطرح، وعندما تتساوى الأفضلية ، يبدأ التقييم من اليسار.
- ✓ تستخدم الأقواس لفرض أفضلية أعلى أو لزيادة وضوح التعبير الحسابي.

أمثلة:

```
SELECT last_name, salary, salary + 300
FROM   employees;
```

```
SELECT last_name, salary, 12*(salary+100)
FROM   employees;
```

ملاحظة: العمود الناتج لن يضاف إلى بنية الجدول، وإنما هو لعرض النتيجة فقط.

القيمة غير المحددة NULL: وهي قيمة غير معروفة ليست صفر وليست فراغ، وإذا كانت قيمة أي طرف من أطراف تعبير حسابي هي NULL فإن النتيجة هي NULL

تعريف اسم مستعار (alias) للعمود:

- ✓ يغير تسمية ترويسة العمود عند عرضه.
- ✓ يتبع اسم العمود مباشرة ، ويمكن استخدام AS لتعريف الاسم المستعار للعمود.
- ✓ يتطلب علامات اقتباس مزدوجة إن احتوى فراغات أو محارف خاصة أو كان حساساً لحالة الأحرف.

```
SELECT last_name  "Name",
       salary*12 "Annual Salary"
FROM   employees;
```

```
SELECT last_name AS name, commission_pct comm
FROM   employees;
```

استخدام عبارة WHERE : تستخدم لفرض شروط على اظهار النتائج في عبارة Select

```
SELECT employee_id, last_name, job_id, department_id
FROM   employees
WHERE  department_id = 90;
```

السلاسل المحرفية والتواريخ (Character Strings and Dates)

- تتم إحاطة قيم السلاسل النصية والتاريخ بعلامة تنصيص مفردة.
- قيم السلاسل النصية حساسة لحالة الأحرف ، وقيم التاريخ حساسة للصيغة .
- الصيغة الافتراضية لقيم التاريخ هي DD-MON-RR .

مثال

```
SELECT last_name, job_id, department_id
FROM   employees
WHERE  last_name = 'Goyal';
```

أمثلة أخرى:

```
... WHERE hire_date='01-JAN-95'
... WHERE salary>=6000
... WHERE last_name='Smith'
```

معاملات المقارنة:

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

Operator	Meaning
BETWEEN ...AND...	Between two values (inclusive)
IN(set)	Match any of a list of values
LIKE	Match a character pattern
IS NULL	Is a null value

أمثلة:

```
SELECT last_name, salary
FROM   employees
WHERE  salary BETWEEN  2500 AND 3500;
```

Lower limit Upper limit

```
SELECT employee_id, last_name, salary, manager_id
FROM   employees
WHERE  manager_id IN (100, 101, 201);
```

```
SELECT employee_id, manager_id, department_id
FROM   employees
WHERE  last_name IN ('Hartstein', 'Vargas');
```

نستخدم **معامل الشرط LIKE** للبحث باستخدام ما يسمى المحارف البديلة (Wildcard Characters) وهي :

- % : ينوب عن أي عدد من المحارف بما فيها العدد صفر من المحارف.
- _ : ينوب عن محرف واحد فقط.

مثال(1):

```
SELECT   first_name
FROM     employees
WHERE    first_name LIKE 'S%';
```

مثال(2) : يوضح إمكانية دمج المحارف البديلة.

```
SELECT last_name
FROM   employees
WHERE  last_name LIKE '_o%';
```

بعد التنفيذ ، سيتم عرض الصفوف الخاصة بالموظفين الذين لديهم الحرف الثاني من الكنية هو "o" .

نستخدم **معامل الشرط IS NULL** للبحث عن الأسطر التي تحوي قيمة NULL .

مثال(1) : استخدام المعامل **IS NULL** لمعرفة اسم مدير الشركة

```
SELECT last_name, manager_id
FROM   employees
WHERE  manager_id IS NULL;
```

مثال(2) : استخدام المعامل **IS NULL** لمعرفة الموظفين الذين لا يأخذون نسبة عمولة (commission percentage) :

```
SELECT last_name, job_id, commission_pct
FROM   employees
WHERE  commission_pct IS NULL;
```

المعاملات المنطقية(Logical Operators)

```
SELECT employee_id, last_name, job_id, salary
FROM   employees
WHERE  salary >=10000
AND    job_id LIKE '%MAN%';
```

```
SELECT employee_id, last_name, job_id, salary
FROM   employees
WHERE  salary >= 10000
OR     job_id LIKE '%MAN%';
```

```
SELECT last_name, job_id
FROM   employees
WHERE  job_id NOT IN ('IT_PROG', 'ST_CLERK', 'SA_REP');
```

ملاحظة:

يمكن استخدام المعامل NOT مع معاملات SQL أخرى كما في الأمثلة التالية:

```
... WHERE  job_id    NOT  IN ('AC_ACCOUNT', 'AD_VP')
... WHERE  salary    NOT  BETWEEN  10000 AND  15000
... WHERE  last_name NOT  LIKE '%A%'
... WHERE  commission_pct  IS   NOT  NULL
```

قواعد الأولوية:

Order Evaluated	Operator
1	Arithmetic operators
2	Concatenation operator
3	Comparison conditions
4	IS [NOT] NULL, LIKE, [NOT] IN
5	[NOT] BETWEEN
6	NOT logical condition
7	AND logical condition
8	OR logical condition

يمكن التغلب على قواعد الأسبقية باستخدام الأقواس.

مثال (1) : معامل AND أقوى من OR.

```
SELECT last_name, job_id, salary
FROM   employees
WHERE  job_id = 'SA_REP'
OR     job_id = 'AD_PRES'
AND    salary > 15000;
```

مثال (2) : استخدام الأقواس لفرض الأفضلية التي نريدها.

```
SELECT last_name, job_id, salary
FROM   employees
WHERE  (job_id = 'SA_REP'
OR     job_id = 'AD_PRES')
AND    salary > 15000;
```

عبارة ORDER BY

- يمكن ترتيب الصفوف المعروضة (وليس الأصلية) باستخدام جملة **ORDER BY** .
  - **ASC :** الترتيب تصاعدي Ascending (وهو الحالة الافتراضية)
  - **DESC :** الترتيب تنازلي Descending
- تأتي جملة **ORDER BY** في نهاية عبارة SELECT .

مثال: ترتيب الموظفين حسب تاريخ التوظيف (الترتيب الافتراضي تصاعدي أي الموظف الأقدم أولاً).

```
SELECT   last_name, job_id, department_id, hire_date
FROM     employees
ORDER BY hire_date;
```

مثال : عكس المثال السابق.

```
SELECT   last_name, job_id, department_id, hire_date
FROM     employees
ORDER BY hire_date DESC;
```

مثال : يرتب حسب الاسم المستعار للعمود.

```
SELECT employee_id, last_name, salary*12 annsal
FROM   employees
ORDER BY annsal;
```

مثال : الترتيب وفق عدة أعمدة.

```
SELECT last_name, department_id, salary
FROM   employees
ORDER BY department_id, salary DESC;
```

هنا يثبت رقم القسم ، ويرتب الرواتب فيه .
ويجب الانتباه إلى أنه بالإمكان الترتيب حسب عمود ليس مذكوراً في لائحة SELECT .

DISTINCT: إن العرض الافتراضي يحوي جميع أسطر النتيجة حتى لو كانت مكررة ، نستطيع التخلص من الأسطر المكررة باستخدام الكلمة المحجوزة DISTINCT في عبارة SELECT

```
SELECT DISTINCT department_id
FROM   employees;
```

التوابع (Functions):

- التوابع السطرية (Single-row functions) : تطبق على كل سطر ، وتعيد قيمة وحيدة لكل سطر.
- توابع التجميع (Multiple-row functions): تطبق على مجموعة من الأسطر، وتعيد قيمة وحيدة من أجل المجموعة.

أولاً: Single-row functions: من هذه التوابع:

- Character functions: تطبق على البيانات النصية وتعيد قيمة نصية أو رقم.
- Number functions: تطبق على البيانات العددية وتعيد قيمة عددية.
- Date functions: تطبق على البيانات من نوع تاريخ / وقت (Date).

1. Character Functions:

Function	Purpose
LOWER(*column|expression*)	Converts alpha character values to lowercase
UPPER(*column|expression*)	Converts alpha character values to uppercase
CONCAT(*column1|expression1*, *column2|expression2*)	Concatenates the first character value to the second character value; equivalent to concatenation operator (||)

مثال1:

```
SELECT  lower(first_name), upper(last_name)
FROM    employees;
```

* Group Functions:

تعمل هذه التوابع على مجموعات من الصفوف وتعيد قيمة وحيدة لكل مجموعة.

وهذه التوابع هي:

- SUM, AVG
- MAX, MIN
- COUNT

ملاحظات:

- استخدام DISTINCT يجعل التابع يأخذ بعين الاعتبار القيم غير المكررة فقط، بينما في الحالة الافتراضية سيتم الأخذ بعين الاعتبار كل قيمة موجودة.
- الوسيط expr يمكن أن يكون من النوع Char, Varchar, Number, Date.
- جميع التوابع السابقة تتجاهل القيم غير المعرفة (Null Values).
- يمكن استخدام التابعين MAX, MIN مع أي نوع من أنواع البيانات.
- التوابعِ AVG, SUM تستخدم فقط مع البيانات الرقمية.

أمثلة:

```
SELECT AVG(salary), MAX(salary),
       MIN(salary), SUM(salary)
FROM   employees
WHERE  job_id LIKE '%REP%';
```

```
SELECT MIN(hire_date), MAX(hire_date)
FROM   employees;
```

```
SELECT  MIN(last_name) , MAX(last_name)
FROM    employees;
```

```
SELECT COUNT(*)
FROM   employees
WHERE  department_id = 50;
```

يعيد التابع (*) COUNT عدد الأسطر في الجدول، بما في ذلك القيم المكررة وقيم NULL.

أما التابع (expr) COUNT فهو يعيد عدد الأسطر متجاهلاً قيم NULL.

والتابع (DISTINCT expr) COUNT يعيد عدد القيم غير المكررة متجاهلاً قيم NULL.

```
SELECT COUNT(commission_pct)
FROM employees
WHERE department_id = 80;
```

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

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

إنشاء مجموعات من البيانات:

يمكن تقسيم الجدول إلى مجموعات صغيرة، وتطبيق التوابع السابقة الذكر على تلك المجموعات، وذلك باستخدام عبارة GROUP BY التي تحدد كيف سيتم تجميع الصفوف.

الشكل العام لاستخدام تلك العبارة:

```
SELECT        column, group_function(column)
FROM          table
[WHERE        condition]
[GROUP BY     group_by_expression]
[ORDER BY     column];
```

ملاحظات:

1. عند تضمين عبارة SELECT أي من التوابع السابقة الذكر، فإن أي عمود آخر سيتم ذكره في عبارة SELECT يجب أن يذكر في عبارة GROUP BY (أي سيتم تجميع الصفوف حسب هذه الأعمدة) ، وإلا سنحصل على رسالة خطأ.
2. استخدام عبارة WHERE تساعد على استبعاد الأسطر التي لا تحقق الشرط قبل تقسيم الجدول إلى مجموعات.
3. لا يمكن استخدام أسماء مستعارة للأعمدة في عبارة GROUP BY.

مثال: نريد حساب المتوسط الحسابي لرواتب كل قسم.

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

يجب الإشارة إلى أنه ليس من الضرورة أن الأعمدة المذكورة في عبارة GROUP BY ستكون موجودة في عبارة SELECT.

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

يمكن استخدام التوابع السابقة في عبارة ORDER BY

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

لا يمكن استخدام التوابع السابقة في عبارة WHERE ، المثال التالي سيعطي رسالة خطأ عند تنفيذه:

```
SELECT      department_id , AVG (salary)
FROM        employees
WHERE       AVG (salary)  > 8000
GROUP BY    department_id;
```

وإنما عند الحاجة لتطبيق شرط معين على توابع التجميع فإننا نستخدم عبارة جديدة هي HAVING، نضع فيها الشروط المراد تطبيقها على توابع التجميع.

```
SELECT department_id, AVG(salary)
FROM   employees
HAVING  AVG(salary) > 8000
GROUP BY department_id;
```

استخدام توابع التجميع المتداخلة:

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