MySQL,
The Comprehensive Course
الدورة الشاملة

إعداد وتقديم: د. عبدالناصر ضياف

07

الربط العمودي والربط الأفقي والجداول الظاهرية

Vertical Join, Horizontal Join, and Virtual Tables

- نتعرف هنا على
- الربط العمودي بين الجداول أو العلاقات بالاعتماد على السجلات أو الصفوف
- الربط الأفقي بين الجداول أو العلاقات بالاعتماد على السمات أو الأعمدة
- كيفية بناء المشاهد Views

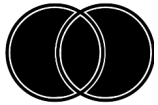
ربط مخرجات الجداول وبناء المشاهد

Joining Tables & Relational Views

- عمليات الفئات Set Operations
- ربط الجداول Joins
- الجداول الظاهرية Virtual Tables وبناء المشاهد Views

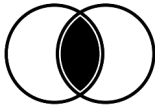
عمليات الفئات Set Operations

- الاتحاد UNION



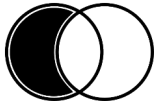
  - تقوم بدمج صفوف مخرجات استفسارين
  - `<query1> UNION [ALL | DISTINCT] <query2>`

- التقاطع INTERSECT



  - تقوم باستخلاص الصفوف المشتركة بين مخرجات استفسارين
  - `<query1> INTERSECT [ALL | DISTINCT] <query2>`

- الفرْق EXCEPT



  - تقوم بحذف الصفوف من مخرجات الاستفسار الأول التي توجد بالاستفسار الثاني
  - `<query1> EXCEPT [ALL | DISTINCT] <query2>`

عمليات الفئات Set Operations

- **شرط**: لابد من تطابق الأعمدة المتناظرة نوعًا ونطاقًا
  - يُفضل اختيار أسماء موحدة للأعمدة وإلا يتم تبني أسماء العلاقة الأولى
  - في حال أردنا الترتيب فيجب استخدام أسماء أعمدة العلاقة الأولى مع فقرة `ORDER BY`

- العمليات الثلاث تقوم تلقائيًا بالتخلص من الصفوف المكررة `DISTINCT` أما في حال الرغبة في إظهار التكرار يستوجب استخدام الفقرة الملحقة `ALL`

- استجلاب قائمة بأسماء وألقاب جميع الأفراد بصرف النظر عن كونهم موظفين أو زبائن

```
SELECT firstName, lastName FROM employees
UNION
SELECT contactFirstName, contactLastName FROM customers;
```

- جرب إعادة تنفيذها باستخدام `UNION DISTINCT`
  - هل تغيرت النتيجة؟

عمليات الفئات Set Operations

- ماذا لو كنا نريد الاسم بالكامل

```
SELECT CONCAT(firstName, ' ', lastName) AS fullName FROM employees
UNION
SELECT CONCAT(contactFirstName, ' ', contactLastName) FROM customers;
```

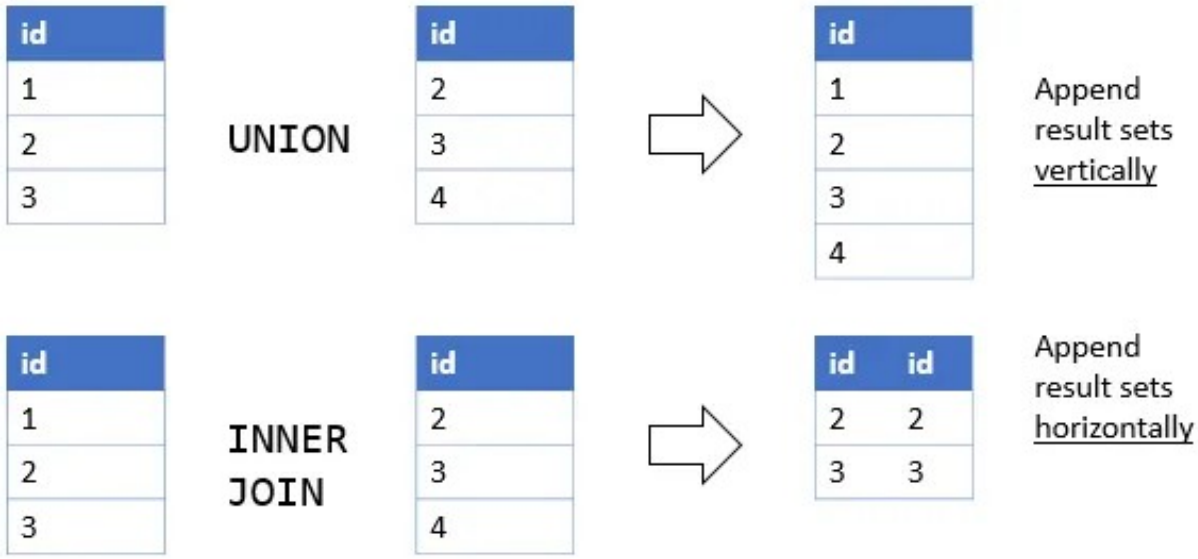
- استجلاب قائمة بالأسماء الأولى للموظفين الغير مشتركة مع مندوبي الزبائن

```
SELECT firstName FROM employees
EXCEPT
SELECT contactFirstName FROM customers;
```

- ماذا لو كنا نريد القائمة بما في ذلك التكرار

```
SELECT firstName FROM employees
EXCEPT ALL
SELECT contactFirstName FROM customers;
```

من الربط العمودي إلى الربط الأفقي



ربط الجداول في SQL

Joining Tables in SQL

- الربط الطبيعي Natural or Inner Join
- الربط الخارجي Outer Join
  - الربط الخارجي الأيمن Right Outer Join
  - الربط الخارجي الأيسر Left Outer Join
  - الربط الخارجي الكلي Full Outer Join
- الربط التقاطعي Cross Join
- الربط الذاتي أو الربط المعاود Self or Recursive Join

ربط جداول قاعدة البيانات Joining Database Tables

- قد تكون القدرة على دمج combine أو ربط join الجداول ذات السمات المشتركة هي من أهم الخصائص التي تتميز بها قواعد البيانات العلائقية على غيرها من الأنواع
- يتم الربط عند الحاجة لاستخلاص البيانات من أكثر من جدول في نفس الوقت
- يتم الربط من خلال تعليمة SELECT حيث
  - يتم سرد قائمة الجداول المترابطة في عبارة FROM
  - تُستخدم عبارة WHERE لتبيان السمات المشتركة والمستخدمة في ربط الجداول
    - تسمى عبارة WHERE هنا بشرط الربط *Join Condition*
- شرط الربط *Join Condition* يحتوي **عادة** على مقارنة مطابقة بين المفتاح الأجنبي Foreign Key بجدول ما والمفتاح الرئيسي Primary Key للجدول المرجعي.

ربط جداول قاعدة البيانات Joining Database Tables

- **تذكر:** عندما يكون لديك ثلاثة جداول أو أكثر في عملية الربط ستحتاج لتحديد شرط ربط لكل زوج من الجداول
  - عدد شروط الربط تكون دائمًا **N-1** حيث N تمثل عدد الجداول المشاركة في عملية الربط

- **مثال (Sakila db):**
  - عرض بيانات استئجار الأفلام على النحو التالي: تاريخ الإيجار – اسم أو لقب الزبون – عنوان الفلم، على أن تكون مرتبة حسب تاريخ الإستئجار ثم اسم الزبون

```
SELECT rental_date, last_name, title
FROM rental, customer, inventory, film
WHERE rental.customer_id=customer.customer_id
    AND rental.inventory_id=inventory.inventory_id
    AND inventory.film_id=film.film_id
ORDER BY rental_date, last_name
```

- **انتبه:** تجنب استخدام شروط الربط الدائرية Circular Join Conditions

ربط جداول قاعدة البيانات Joining Database Tables

- **تذكر:** عندما يتشابه أسماء السمات المطلوبة في أكثر من جدول يستوجب أن تُسبق السمة باسم الجدول التابعة له (مثل: customer.id)

- **مثال (Sakila db):**
  - عرض بيانات استئجار الأفلام على النحو التالي: تاريخ الإيجار – اسم أو لقب الزبون – عنوان الفلم، اسم أو لقب الموظف المسؤول، على أن تكون مرتبة حسب تاريخ الإستئجار ثم اسم الزبون

```
SELECT rental_date, customer.last_name, title, staff.last_name
FROM rental, customer, staff, inventory, film
WHERE rental.customer_id=customer.customer_id
    AND rental.inventory_id=inventory.inventory_id
    AND inventory.film_id=film.film_id
    AND rental.staff_id=staff.staff_id
ORDER BY rental_date, customer.last_name
```

- **ماذا لو** تشابهت أسماء سمات العلاقة الناتجة كما في المثال السابق (*last_name*)!!
  - كيف سنميزها إذًا؟

ربط الجداول باستخدام الأسماء المستعارة Aliases

- يمكن استخدام الأسماء المستعارة Aliases على مستوى الجداول tables أو على مستوى السمات attributes بهدف تجنب التشابه في الأسماء أو اختصار أسمائها
- **تحسين المثال السابق**

```
SELECT r.rental_date, c.last_name AS customer, f.title,
   s.last_name AS staff
FROM rental r, customer c, staff s, inventory i, film f
WHERE r.customer_id=c.customer_id
    AND r.inventory_id=i.inventory_id
    AND i.film_id=f.film_id
    AND r.staff_id=s.staff_id
ORDER BY r.rental_date, c.last_name
```

الروابط الذاتية للجداول Recursive Joins

- يعتبر الاسم المستعار Alias مفيدًا بشكل خاص وإجباريًا عند الحاجة لربط الجدول بنفسه في استفسار معاود وذلك لتمييز الجدول عن نفسه والتعامل مع كل اسم مستعار كعلاقة مستقلة

- **مثال:** استجلاب قائمة ببيانات الموظفين تحتوي رقم الموظف واسمه بالكامل واسم رئيسه المباشر

```
SELECT e.employeeNumber,
  CONCAT(e.firstName, ' ', e.lastName) employeeName,
  CONCAT(m.firstName, ' ', m.lastName) managerName
FROM employees e, employees m
WHERE e.reportsTo = m.employeeNumber
ORDER BY m.employeeNumber
```

- دعنا نجرّب التالي ونرى الفرق:

```
table1 INNER JOIN table2 ON join_condition  -
      table1 JOIN table2 ON join_condition  -
```

الروابط الخارجية Outer Joins

- توفر SQL نوعان من الروابط الخارجية: يمنى (Right) ويسرى (Left)
- **الرابط الخارجي الأيمن *Right outer join*** يقوم بربط كلا الجدولين بحيث يعرض جميع صفوف الجدول الأيمن مع الصفوف المحققة للمطابقة من الجدول الأيسر.

```
SELECT column_list
FROM left_table_name RIGHT JOIN right_table_name
ON join_condition;
```

- **الرابط الخارجي الأيسر *Left outer join*** يقوم بربط كلا الجدولين بحيث يعرض جميع صفوف الجدول الأيسر مع الصفوف المحققة للمطابقة من الجدول الأيمن

```
SELECT column_list
FROM left_table_name LEFT JOIN right_table_name
ON join_condition;
```

الروابط الخارجية Outer Joins

- هل قائمة الموظفين كانت كاملة في مخرجات المثال السابق؟!
  - فلنقم بتبديل `LEFT JOIN` إلى `INNER JOIN`

```
SELECT e.employeeNumber,
  CONCAT(e.firstName, ' ', e.lastName) employeeName,
  CONCAT(m.firstName, ' ', m.lastName) managerName
FROM employees e LEFT JOIN employees m
ON e.reportsTo = m.employeeNumber
ORDER BY m.employeeNumber
```

- المطلوب استجلاب قائمة بأسماء جميع الزبائن دون استثناء وحجم معاملاتهم المالية

```
SELECT customerNumber, customerName, SUM(amount) total
FROM customers c LEFT JOIN payments p USING (customerNumber)
GROUP BY customerNumber
ORDER BY total DESC
```

- الاختصار والوضوح والتقيد لـ USING بينما المرونة والتعدد والتنوع لـ ON

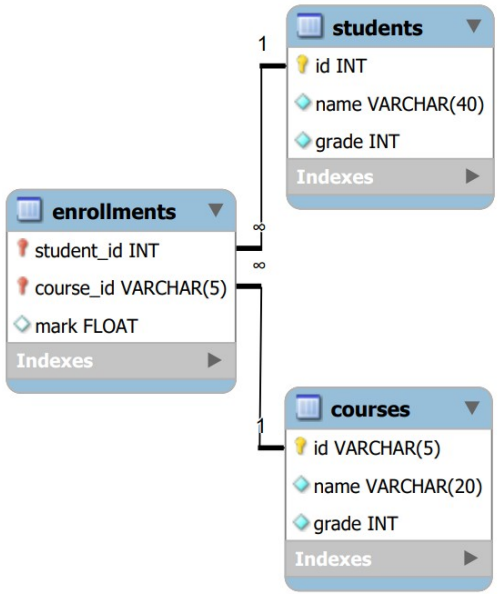
الروابط الخارجية Outer Joins

- أين الربط الخارجي الكلي Full Outer Join
  - MySQL لم تقدم حتى الآن هذه العملية بشكل مباشر بسبب توفرها وبكل بساطة من خلال اتحاد مخرجات كل من الربط الخارجي الأيسر مع مخرجات الربط الخارجي الأيمن
  - دعونا نعرض جميع الموظفين كمرؤوسين مع رؤساؤهم وكرؤساء مع مرؤوسيهم

```
SELECT CONCAT(e.firstName, ' ', e.lastName) employeeName,
  CONCAT(m.firstName, ' ', m.lastName) managerName
FROM employees e LEFT JOIN employees m ON e.reportsTo = m.employeeNumber
UNION
SELECT CONCAT(e.firstName, ' ', e.lastName) employeeName,
  CONCAT(m.firstName, ' ', m.lastName) managerName
FROM employees e RIGHT JOIN employees m ON e.reportsTo = m.employeeNumber
```

- الربط التقاطعي CROSS JOIN
  - فلنقم ببناء قاعدة بيانات لنموذج تنزيل مقررات الفصل المغلق للطلاب ونختبر فيها الربط التقاطعي والإدخال من مخرجات استفسار

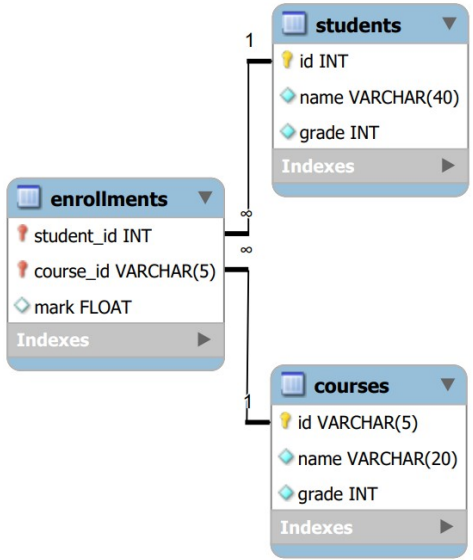
الربط التقاطعي Cross Join




```
CREATE DATABASE my_school;
USE my_school;

CREATE TABLE students (
  id int NOT NULL PRIMARY KEY,
  name varchar(40) NOT NULL,
  grade int NOT NULL
);

CREATE TABLE courses (
  id varchar(4) NOT NULL PRIMARY KEY,
  name varchar(20) NOT NULL,
  grade int NOT NULL
);

CREATE TABLE enrollments (
  student_id int NOT NULL,
  course_id varchar(5) NOT NULL,
  mark float,
  PRIMARY KEY (student_id, course_id),
  FOREIGN KEY (student_id) REFERENCES students(id),
  FOREIGN KEY (course_id) REFERENCES courses(id)
);
```

الربط التقاطعي Cross Join

```
INSERT INTO students VALUES
(23001, '2', 'أحمد عبدالله'),
(23002, '2', 'علياء محمد'),
(23003, '4', 'محمد فراس'),
(23004, '6', 'محمود علي'),
(23005, '4', 'سعاد احميده');

INSERT INTO courses VALUES
('G201', '2', 'حساب'),
('G202', '2', 'قراءة'),
('G401', '4', 'رياضيات'),
('G402', '4', 'مطالعة'),
('G601', '6', 'أحياء'),
('G602', '6', 'كيمياء'),
('G703', '6', 'علوم');
```




```
INSERT INTO enrollments (student_id, course_id)
  SELECT s.id, c.id FROM students s CROSS JOIN courses c USING (grade);
```

- هل هذا فعليًا ربط تقاطعي؟! وماذا تفعل `USING` هنا؟!

الجداول الظاهرية وبناء المشاهد
Virtual Tables and Creating Views

- **تذكر:** مخرجات كل تعليمة استفسار SELECT تتم على قاعدة بيانات علائقية RDB هي عبارة عن علاقة جديدة Relation (أو جدول ظاهري Virtual Table)

- ماذا لو كان عليك عند نهاية كل يوم الحصول على قائمة بالأصناف التي وصل مخزونها إلى الحد الأدنى بهدف إعداد طلبية بها،

  - فبدلاً من تحرير نفس الاستفسار query يوميًا، أليس من الأفضل حفظ الاستفسار داخل قاعدة البيانات بشكل دائم؟!

  - هذه بالضبط وظيفة المشهد العلائقي Relational View

الجداول الظاهرية وبناء المشاهد

Virtual Tables and Creating Views

- **المشهد View**: هو جدول ظاهري ينتج عن تعليمة استفسار SELECT
- يمكن أن تحتوي تعليمة SELECT لبناء المشهد على
  - أعمدة حقيقية Columns
  - أعمدة محسوبة Computed Columns
  - أسماء مستعارة Aliases
  - دوال مجاميع Aggregate Functions

  من جدول واحد أو أكثر.
- **الجداول الأساس Base Tables**: هي الجداول التي يعتمد عليها المشهد.
- يتم بناء المشهد من خلال التعليمة:

```
CREATE VIEW view_name AS SELECT_query
```

الجداول الظاهرية وبناء المشاهد
Virtual Tables and Creating Views

- **تذكر:** بالرغم من أن تعليمة CREATE تعتبر من ضمن أوامر تعريف البيانات DDL إلا أن وظيفتها تتوافق مع أوامر معالجة البيانات DML باعتبارها انعكاس لتعليمة الاستفسار SELECT

- **خصائص المشهد العلائقي:**
  - يمكن استخدام اسم المشهد في أي استفسار query كما لو كان جدولاً
  - يتم تحديث محتويات المشهد ديناميكيًا عند كل تعامل معه
  - تمنح المشاهد درجة إضافية لأمن قاعدة البيانات DB Security باعتبارها تقيّد المستخدم بأعمدة معينة وصفوف معينة
  - يمكن أن تُستخدم المشاهد كأساس للعديد من التقارير

- **تذكر:** يمكننا مبدئيًا اعتبار أن المشاهد هي للقراءة فقط Read-Only إلا أنها يمكن أن تكون قابلة للتعديل إذا حققت شروط معينة (ندرسها لاحقا).

الجداول الظاهرية وبناء المشاهد
Virtual Tables and Creating Views

- المطلوب إعداد مشهد لاستجلاب قائمة بأسماء جميع الزبائن دون استثناء وحجم معاملاتهم المالية

```
CREATE VIEW customer_payment_subtotals AS
SELECT customerNumber, customerName, SUM(amount) total
FROM customers c LEFT JOIN payments p USING
  (customerNumber)
GROUP BY customerNumber
```

- هل يمكن استخدام المشهد لإنتاج قائمة من الثوابت؟!
  - شهور السنة مثلاً

الجداول الظاهرية وبناء المشاهد

Virtual Tables and Creating Views

```
SELECT 'JAN' AS month_name
 UNION SELECT 'FEB'
 UNION SELECT 'MAR'
 UNION SELECT 'APR'
 UNION SELECT 'MAY'
 UNION SELECT 'JUN'
 UNION SELECT 'JUL'
 UNION SELECT 'AUG'
 UNION SELECT 'SEP'
 UNION SELECT 'OCT'
 UNION SELECT 'NOV'
 UNION SELECT 'DEC';
```

```
+------------+
| month_name |
+------------+
| JAN        |
| FEB        |
| MAR        |
| APR        |
| MAY        |
| JUN        |
| JUL        |
| AUG        |
| SEP        |
| OCT        |
| NOV        |
| DEC        |
+------------+
```