# SQL (2)

# Tabel Awal: Products

| ProductID | ProductName | Price | Category |
|---|---|---|---|
| 1 | Apple | 1.00 | Fruits |
| 2 | Banana | 0.50 | Fruits |
| 3 | Orange | 0.75 | Fruits |
| 4 | Mango | 1.50 | Fruits |
| 5 | Pineapple | 1.25 | Fruits |
| 6 | Carrot | 0.30 | Vegetables |
| 7 | Broccoli | 0.45 | Vegetables |

# SQL Query LIMIT

## Contoh Query

```
SELECT * FROM Products LIMIT 5;
```

## Hasil Query

| ProductID | ProductName |
| --- | --- |
| 1 | Apple |
| 2 | Banana |
| 3 | Orange |
| 4 | Mango |
| 5 | Pineapple |

# SQL Query OFFSET

```sql
SELECT * FROM Products ORDER BY ProductID LIMIT 3 OFFSET 2;
```

- **Definisi**: OFFSET digunakan untuk melewati sejumlah baris sebelum memulai pengambilan baris dalam query.
- **Kegunaan**: Berguna untuk implementasi pagination.

# SQL Query SELECT

## Contoh Query

```
SELECT ProductName, Price FROM Products;
```

## Hasil Query

| ProductName | Price |
| --- | --- |
| Apple | 1.00 |
| Banana | 0.50 |
| Orange | 0.75 |

# SQL Query WHERE

## Contoh Query

```
SELECT * FROM Products WHERE Price > 1.00;
```

## Hasil Query

| ProductID | ProductName | Price |
|---|---|---|
| 4 | Mango | 1.50 |
| 5 | Pineapple | 1.25 |

# SQL Query GROUP BY

## Contoh Query

```
SELECT Category, COUNT(*) FROM Products GROUP BY Category;
```

## Hasil Query

| Category | COUNT(*) |
| --- | --- |
| Fruits | 5 |
| Vegetables | 2 |

# SQL Query ORDER BY

## Contoh Query

```
SELECT * FROM Products ORDER BY Price DESC;
```

## Hasil Query

| ProductID | ProductName | Price |
|---|---|---|
| 4 | Mango | 1.50 |
| 5 | Pineapple | 1.25 |
| 1 | Apple | 1.00 |

# SQL Query COUNT

```sql
SELECT COUNT(*) FROM Products WHERE Category = 'Fruits';
```

- **Definisi**: COUNT adalah fungsi agregat yang menghitung jumlah baris yang sesuai dengan kriteria tertentu.

# SQL Query SUM dan AVG

```sql
-- Menghitung total dan rata-rata harga buah
SELECT SUM(Price) FROM Products WHERE Category = 'Fruits';
SELECT AVG(Price) FROM Products WHERE Category = 'Fruits';
```

- **SUM**: Menghitung total dari kolom numerik.
- **AVG**: Menghitung rata-rata dari kolom numerik.

# SQL Query BETWEEN, GREATER/LESS THAN

```
SELECT * FROM Products WHERE Price BETWEEN 0.50 AND 1.00;
SELECT * FROM Products WHERE Price > 1.00;
SELECT * FROM Products WHERE Price < 0.50;
```

- **BETWEEN**: Memilih baris di mana nilai kolom berada dalam rentang tertentu.
- **GREATER THAN (>)**: Memilih baris dengan nilai lebih besar dari nilai yang ditentukan.
- **LESS THAN (<)**: Memilih baris dengan nilai lebih kecil dari nilai yang ditentukan.

# CREATE TABLE dengan Primary dan Foreign Key

## Contoh Query

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

Tabel baru **Orders** telah dibuat dengan `OrderID` sebagai Primary Key dan `ProductID` sebagai Foreign Key.

# CREATE TABLE dengan Primary dan Foreign Key

## Tabel Orders

| OrderID | OrderNumber | ProductID |
|---|---|---|
| 1 | 1001 | 4 |
| 2 | 1002 | 2 |
| 3 | 1003 | 1 |

# SQL Query JOIN

- **INNER JOIN** Menggabungkan baris dari dua tabel ketika kondisi join terpenuhi.
- **LEFT JOIN (LEFT OUTER JOIN)** Mengembalikan semua baris dari tabel kiri dan baris yang cocok dari tabel kanan.
- **RIGHT JOIN (RIGHT OUTER JOIN)** Mengembalikan semua baris dari tabel kanan dan baris yang cocok dari tabel kiri.
- **FULL JOIN (FULL OUTER JOIN)** Menggabungkan hasil dari LEFT JOIN dan RIGHT JOIN.
- **CROSS JOIN** Menghasilkan Cartesian Product antara dua tabel.

# SQL Query JOIN: Base Tables

**Tabel Students**: Menyimpan informasi tentang student.

| StudentID | StudentName |
|---|---|
| 1 | Alice |
| 2 | Bob |
| 3 | Charlie |
| 4 | David |

# SQL Query JOIN: Base Tables

**Tabel Enrollments**: Menyimpan informasi tentang pendaftaran student di course.

| EnrollmentID | StudentID | CourseID |
|---|---|---|
| E001 | 1 | C101 |
| E002 | 2 | C102 |
| E003 | 3 | C103 |
| E004 | 1 | C104 |

# SQL Query JOIN: Base Tables

**Tabel Courses**: Menyimpan informasi tentang course yang ditawarkan.

| CourseID | CourseName |
|---|---|
| C101 | Math |
| C102 | Science |
| C103 | History |
| C104 | Literature |
| C105 | Geography |

# SQL INNER JOIN

## Definisi

`INNER JOIN` adalah operator SQL yang menggabungkan baris dari dua tabel berdasarkan nilai yang cocok pada kolom yang sama.

## Kegunaan

INNER JOIN digunakan untuk mengambil baris yang memiliki nilai yang cocok di kedua tabel.

# SQL INNER JOIN: Syntax

```
SELECT
  Students.StudentName,
  Courses.CourseName
FROM
  Enrollments
INNER JOIN Students ON Enrollments.StudentID = Students.StudentID
INNER JOIN Courses ON Enrollments.CourseID = Courses.CourseID;
```

# SQL INNER JOIN: Result

| StudentName | CourseName |
| --- | --- |
| Alice | Math |
| Bob | Science |
| Charlie | History |
| Alice | Literature |

# SQL INNER JOIN

**Bagaimana INNER JOIN Bekerja**:

- `INNER JOIN` membuat kombinasi baris antara dua tabel ketika kondisi yang ditentukan terpenuhi.
- Hanya baris yang cocok di kedua tabel yang akan ditampilkan di hasil akhir.

**Proses Join**

1. Pertama, `JOIN` dilakukan antara `Enrollments` dan `Students` menggunakan `StudentID`.
2. Kemudian, hasilnya di-`JOIN` dengan tabel `Courses` menggunakan `CourseID`.
3. Akhirnya, data yang cocok dari ketiga tabel digabungkan menjadi satu hasil tabel.

# SQL LEFT JOIN

**Definisi**: `LEFT JOIN` (juga dikenal sebagai `LEFT OUTER JOIN`) mengembalikan semua baris dari tabel kiri (tabel pertama) dan baris yang cocok dari tabel kanan (tabel kedua). Jika tidak ada kecocokan, hasilnya adalah `NULL` di sisi kanan.

# SQL LEFT JOIN: Syntax

```
SELECT
  Students.StudentName,
  Courses.CourseName
FROM
  Students
LEFT JOIN Enrollments ON Students.StudentID = Enrollments.StudentID
LEFT JOIN Courses ON Enrollments.CourseID = Courses.CourseID;
```

# SQL LEFT JOIN: Result

| StudentName | CourseName |
| --- | --- |
| Alice | Math |
| Bob | Science |
| Charlie | History |
| Alice | Literature |
| David | `NULL` |

# SQL LEFT JOIN

**Bagaimana LEFT JOIN Bekerja**:

- `LEFT JOIN` memastikan bahwa semua baris dari tabel pertama (kiri) muncul dalam hasil, terlepas dari apakah mereka memiliki kecocokan di tabel kedua.
- Jika tidak ada kecocokan di tabel kedua, kolom dari tabel kedua akan diisi dengan `NULL`.

# SQL LEFT JOIN

### Proses Join

1. Setiap baris dari tabel `Students` dipertimbangkan.
2. Baris tersebut di-`JOIN` dengan tabel `Enrollments`. Jika tidak ada kecocokan, kolom `Enrollments` diisi dengan `NULL`.
3. Kemudian, hasil ini di-`JOIN` dengan `Courses`. Jika tidak ada kecocokan, kolom `Courses` diisi dengan `NULL`.
4. Hasilnya adalah daftar lengkap student dengan course yang mereka ikuti, termasuk student yang tidak mengikuti course apa pun.

# SQL LEFT JOIN

## Kasus Penggunaan

Berguna untuk situasi di mana kita ingin menampilkan semua item dari satu set (misalnya, semua student), tanpa memandang apakah ada kecocokan di set lain (misalnya, pendaftaran course).

> LEFT JOIN sering digunakan untuk menemukan kekosongan dalam data, seperti menampilkan semua student, termasuk mereka yang tidak terdaftar dalam course apa pun.

# SQL RIGHT JOIN

**Definisi**: `RIGHT JOIN` (juga dikenal sebagai `RIGHT OUTER JOIN`) mengembalikan semua baris dari tabel kanan (tabel kedua) dan baris yang cocok dari tabel kiri (tabel pertama). Jika tidak ada kecocokan, hasilnya adalah `NULL` di sisi kiri.

# SQL RIGHT JOIN: Syntax

```
SELECT
  Students.StudentName,
  Courses.CourseName
FROM
  Students
RIGHT JOIN Enrollments ON Students.StudentID = Enrollments.StudentID
RIGHT JOIN Courses ON Enrollments.CourseID = Courses.CourseID;
```

# SQL RIGHT JOIN: Result

| StudentName | CourseName |
| --- | --- |
| Alice | Math |
| Bob | Science |
| Charlie | History |
| Alice | Literature |
| `NULL` | Geography |

# SQL FULL JOIN

**Definisi**: `FULL JOIN` (juga dikenal sebagai `FULL OUTER JOIN`) menggabungkan hasil dari `LEFT JOIN` dan `RIGHT JOIN`. Ini mengembalikan semua baris dari kedua tabel, dengan baris dari satu sisi yang tidak memiliki kecocokan di sisi lain diisi dengan `NULL`.

# SQL FULL JOIN: Syntax

```
SELECT
  Students.StudentName,
  Courses.CourseName
FROM
  Students
FULL JOIN Enrollments ON Students.StudentID = Enrollments.StudentID
FULL JOIN Courses ON Enrollments.CourseID = Courses.CourseID;
```

# SQL FULL JOIN: Result

| StudentName | CourseName |
| --- | --- |
| Alice | Math |
| Bob | Science |
| Charlie | History |
| Alice | Literature |
| David | `NULL` |
| `NULL` | Geography |

# SQL CROSS JOIN

**Definisi**: `CROSS JOIN` menghasilkan produk Kartesian antara dua tabel – yaitu, menggabungkan setiap baris dari tabel pertama dengan setiap baris dari tabel kedua.

> Produk Kartesian terjadi ketika menggabungkan setiap baris dari satu tabel dengan setiap baris dari tabel lain

> Jika tabel pertama memiliki `n` baris dan tabel kedua memiliki `m` baris, hasil produk Kartesian akan memiliki `n` * `m` baris

# SQL CROSS JOIN: Syntax

```
SELECT
  Students.StudentName,
  Courses.CourseName
FROM
  Students
CROSS JOIN Courses;
```

# SQL CROSS JOIN: Result

| StudentName | CourseName |
|---|---|
| Alice | Math |
| Alice | Science |
| Alice | History |
| Alice | Literature |
| Alice | Geography |
| Bob | Math |
| Bob | Science |
| Bob | History |
| ... | ... |

# CROSS JOIN

**Bagaimana CROSS JOIN Bekerja**:

- `CROSS JOIN` menggabungkan setiap baris dari satu tabel dengan setiap baris dari tabel lain tanpa memerlukan kondisi spesifik.
- Hasilnya adalah tabel dengan setiap kombinasi baris dari kedua tabel.

# CROSS JOIN

### Karakteristik CROSS JOIN

- Tidak menggunakan klausa `ON` atau `WHERE` karena tidak ada kondisi spesifik untuk join.
- Jumlah baris di tabel hasil adalah perkalian jumlah baris di kedua tabel sumber.
- Biasanya menghasilkan jumlah baris yang sangat besar.

# CROSS JOIN

### Kasus Penggunaan

- `CROSS JOIN` berguna dalam situasi analitis atau pengujian, seperti ketika kita perlu menggabungkan semua kemungkinan pasangan item dari dua set data.
- Sering digunakan dalam perencanaan skenario atau analisis kombinasi.

> Walaupun `CROSS JOIN` kurang umum digunakan dalam aplikasi database sehari-hari, namun memiliki peranan penting dalam kasus analisis data tertentu.