

Chapter 3: SQL Query

- (Based on Microsoft SQL Server)
- Introduction to Relation Database
- Simple Queries on One Table
- Summarizing Data
- Using JOIN with Tables
- Manipulating Data
- Using Subquery
- Combining Multiple Queries into One

Introduction to Relational Database

- What is Relational Database?
 - Relational Database is a set of tables. Each table keeps information about aspects of one thing such as a customer, an order, or a product.
- Data Model
 - Data Model provides us with information about how data items in the database are interrelated.
 - One convenient way to give an overview of the different tables in a database is by using the class diagram notation from the Unified Modeling Language (UML)

1. Simple Queries on One Table

- Relational Algebra and SQL
- Subset of rows and columns
- SELECTION and PROJECTION in Relational Algebra
- Combining SELECTION and PROJECTION
- Simple SQL SELECT Statement
- Using Alias
- Specifying Condition

Relational Algebra and SQL

- Relational Algebra is used to define the way in which relations (tables) can be operated to manipulate their data. It is used as the basis of SQL for relation database.
- SQL = Structured Query Language
 - Pronounced as S-Q-L or sequel
 - Standardized by ANSI (American National Standard Institute)
 - Supported by modern RDBMSs
 - There are three group of commands
 - DLL = Data Definition Language
 - DML = Data Manipulation Language (DML)
 - Data Control Language (grant, revoke, deny privileges)

1.1. Subset of Rows and Columns

- Subsets of Rows and Columns

ID	Name	Gender	Phone	Birth Place
1	Dara	M	012	Takeo
2	Thida	F	012	Phnom Penh
3	Mara	M	012	Kandal
4	Sokha	M	012	Phnom Penh

Table 1: Full rows and columns of table **Student** (List of Student)

ID	Name	Gender	Phone	Birth Place
2	Thida	F	012	Phnom Penh
4	Sokha	M	012	Phnom Penh

Subset of Rows

Name	BirthPlace
Dara	Takeo
Thida	Phnom Penh
Mara	Kandal
Sokha	Phnom Penh

Subset of Columns

1.2. SELECTION with Relational Algebra

- Selection is to retrieve subset of rows in a table by using *condition*.
- σ (*sigma*) is short-hand for **selection**.

$$\sigma_{\langle \text{condition} \rangle} \langle \text{tablename} \rangle$$

Example: $\sigma_{\text{Gender}='M'}(\text{Student})$

This mean that we select all male students from Student Table.

1.3. PROJECTION with Relational Algebra

- Projection is to retrieve subset of columns by *specify columns* in a table.
- π (π) use for selecting subset of columns called **projection**.

$$\pi_{\langle \text{column list} \rangle} \langle \text{tablename} \rangle$$

Example: $\pi_{\text{[Name], Gender, Phone}} (\text{Student})$

This mean that we get all students from Student Table only column Name, Gender and Phone.

1.4. Combining SELECTION and PROJECTION

- To retrieve the combination of subset of rows and columns we need to use both SELECTION and PROJECTION.
- Use π (π) and σ (σ) together:

$$\pi_{\langle \text{columns list} \rangle} (\sigma_{\langle \text{conditions} \rangle} \langle \text{tablename} \rangle)$$

$\pi_{\text{[Name], Gender, Phone}} (\sigma_{\text{Gender='M'}} (\text{Student}))$

This mean that we retrieve all male students on column Name, Gender and Phone only.

1.5. Simple SQL SELECT Statement

- Syntax

```
SELECT [DISTINCT] <columns list>
FROM <table name>
[WHERE <conditions>]
[GROUP BY <columns list>]
[HAVING <columns list>]
[ORDER BY <columns list> [ASC|DESC]]
```

- Text are placed in <...> referred to column name table name, condition, expression, etc.
- In syntax area, the brackets [...] are used for optional statement.
- In usage area, brackets [...] are use with field name or table name which its name the contain special characters or same to some keyword.
- Column list or table list are separated by comma (,)
- When we use asterisk (*) after SELECT keyword it means that all columns are selected.

EXAMPLE of Simple SQL SELECT Statement

- We have one table named **Student** as below:

ID	Name	Gender	Phone	BirthPlace
1	Dara	M	012	Takeo
2	Thida	F	012	Phnom Penh
3	Mara	M	012	Kandal
4	Sokha	M	012	Phnom Penh
5	Sokean	M	012	Kandal
6	Saovira	M		Phnom Penh
7	Sokhom	M	012	Takeo
8	Sokanha	F	012	Phnom Penh
9	Dalin	F	012	Kandal
10	Monypov	F	012	Takeo

Table 2: Table Student

EXAMPLE of Simple SQL SELECT Statement

- Retrieve all female students from Table Student

```
SELECT *
FROM Student
WHERE Gender='F'
```

ID	Name	Gender	Phone	BirthPlace
2	Thida	F	012	Phnom Penh
8	Sokanha	F	012	Phnom Penh
9	Dalin	F	012	Kandal
10	Monypov	F	012	Takeo

EXAMPLE of Simple SQL SELECT Statement

- Retrieve all students in Student Table with column Name, and Birth Place

```
SELECT [Name], [BirthPlace]
FROM Student
```

Name	BirthPlace
Dara	Takeo
Thida	Phnom Penh
Mara	Kandal
Sokha	Phnom Penh
Sokean	Kandal
Saovira	Phnom Penh
Sokhom	Takeo
Sokanha	Phnom Penh
Dalin	Kandal
Monypov	Takeo

- Field name that contains space, keywords, etc. must place in the bracket (eg. [date], [birth place])

EXAMPLE of Simple SQL SELECT Statement

- Retrieve all male students with column Name, Gender, and Phone

```
SELECT [Name], Gender,
       Phone
FROM Student
WHERE Gender='M'
```

Name	Gender	Phone
Dara	M	012
Thida	F	012
Mara	M	012
Sokha	M	012
Sokean	M	012
Saovira	M	012
Sokhom	M	012
Sokanha	F	012
Dalin	F	012
Monypov	F	012

1.6. Written Relational Algebra in SQL

Query	Relational Algebra	SQL
Retrieve all male students	$\sigma_{\text{Gender}='M'}(\text{Student})$	SELECT * FROM Student WHERE Gender='M'
Retrieve all students with column Name and Phone	$\pi_{\text{Name, Phone}}(\text{Student})$	SELECT [Name], Phone FROM Student
Retrieve all male student with Column Name and Phone	$\pi_{\text{Name, Phone}}(\sigma_{\text{Gender}='M'}(\text{Student}))$	SELECT [Name], Phone FROM Student WHERE Gender='M'

1.7. Using Alias

- Using Alias is to alias name of field or table into other name (shortcut name)

```
SELECT A.Field1 [AS] AliasField1, ...
FROM Table1 [AS] AliasName1, ...
```

- Example

```
SELECT Student.[Name], Student.Phone As [H/P]
FROM Student
```

Or (we can use as below)

```
SELECT S.[Name], S.Phone As [H/P]
FROM Student S
```

Name	H/P
Dara	012
...	...

1.8. Specifying Conditions with SQL SELECT Statement

- Comparison Operators

Operator	Meaning	Example of True Statement
=	Equals	1=1, 'Dara'='Dara'
<	Less than	7<8, 'Ana'<'Thida'
>	Greater than	9>3, 'Vuthy'>'Mara'
<=	Less than or equal	2<=3, 7<=7
>=	Greater than or equal	8>=4, 3>=3
<>	Not equal	4<>5, 'Davy'<>'Pros'
Between	Between two values	BETWEEN 3 AND 10, BETWEEN 'A' AND 'Man'

- Logical Operators

Operator	Meaning	Example
AND	True when both conditions true	Gender='F' AND Phone='012'
OR	True when one conditions is true	Phone='012' OR Phone='092'
NOT	Not this condition	NOT Phone='012'

Specifying Conditions with SQL SELECT Statement

- Retrieve all students in one column (name) whose birth place are not in Phnom Penh.

```
SELECT [Name] FROM Student
WHERE NOT (BirthPlace='Phnom Penh')
```

- Retrieve all female students in three columns (name, gender, birthplace) whose birth place are not in Phnom Penh.

```
SELECT [Name], Gender, BirthPlace FROM Student
WHERE Gender='F' AND NOT (BirthPlace='Phnom Penh')
```

- Retrieve all students in all columns whose birth place are in Takeo, Kandal and Kampot.

```
SELECT * FROM
WHERE BirthPlace IN ('Takeo', 'Kandal', 'Kampot')
```

BirthPlace='Takeo' OR BirthPlace='Kandal' OR BirthPlace='Kampot'

Specifying Conditions with SQL SELECT Statement

- Retrieve all male students whose name start with letter 'S'.

```
SELECT * FROM Student
WHERE Gender='M' AND [Name] LIKE 'S%'
```

In MS-Access we use 'S*'

- Retrieve all students whose name end with 'ra'.

```
SELECT * FROM Student
WHERE [Name] LIKE '%ra'
```

- Retrieve all students whose id number between 2 and 8

```
SELECT * FROM Student
WHERE [ID] BETWEEN 2 AND 8
```

- Retrieve all students whose not have phone number.

```
SELECT * FROM Student
WHERE Phone IS NULL
```

Specifying Conditions with SQL SELECT Statement

- Retrieve all student whose name start with any letters and end with 'ara'.

```
SELECT * FROM Student
WHERE [Name] LIKE '_ara'
```

In MS-Access we use '?ara'

- Retrieve all student whose name start with letter 'S' OR 'M' OR 'D'

```
SELECT * FROM Student
WHERE [Name] LIKE '[SMD]ara'
```

1.9. Managing Duplicate

- In Table 2: Column BirthPlace has Duplicated data and we want to get non-duplicated data (retrieve all birth places of students)

BirthPlace	BirthPlace
Takeo	Takeo
Phnom Penh	Phnom Penh
Kandal	Kandal
Phnom Penh	
Kandal	
Phnom Penh	
Takeo	
Phnom Penh	
Kandal	
Takeo	

SELECT [BirthPlace]
FROM Student

With duplicates

SELECT DISTINCT [BirthPlace]
FROM Student

Without duplicates

1.10. Sorting Retrieved Data

- Retrieve all student with ascending order of student name.

```
SELECT * FROM Student
```

```
ORDER BY [Name] ASC
```

– *Note: ASC is default sorting order (we can omit it), otherwise is DESC.*

- Retrieve all male students sort order descending [birth place] and ascending [name]

```
SELECT * FROM Student
```

```
WHERE Gender='M'
```

```
ORDER BY BirthPlace DESC, [Name]
```

2. Summarizing Data

Barcode	Productname	Date	Qty	Price	Amount
0001	ABC	01/01/10	5	12	60
0002	Angkor	01/01/10	8	10	80
0003	Coca	01/01/10	2	8	16
0001	ABC	02/01/10	3	12	36
0003	Coca	02/01/10	1	8	8
0002	Angkor	03/01/10	10	11	110
0003	Coca	04/01/10	5	8	40
0002	Angkor	05/01/10	3	12	36
0004	Anchor	06/01/10	2	9	18
0005	Pepsi	06/01/10	1	8	8
0001	ABC	07/01/10	2	10	20

Table 3: ProductSold

2.1. Using Aggregate Functions and Grouping Data

- SQL Aggregate Functions

Function	Description
SUM ()	Return the sum of a specified column
AVG ()	Return the average value of a specified column
MIN ()	Return the minimum value in a specified column
MAX ()	Return the maximum value in a specified column
COUNT ()	Return number of rows in a specified column

2.2. Using COUNT() Function

- COUNT()

- How many student in Table 2?

```
SELECT COUNT(*) FROM Student
```

- How many students in each place (Table 2)?

```
SELECT BirthPlace, COUNT(*)
FROM Student
GROUP BY BirthPlace
```

- How many male student (Table 2)?

```
SELECT COUNT(*)
FROM Student
WHERE Gender='M'
```

2.3. Using SUM() Function

- SUM ()

- What is the amount of sale from '01/01/10' to '07/01/10' (Table 3)?

```
SELECT SUM(Amount) As Amount
FROM ProductSold
WHERE [Date] BETWEEN '01/01/10' AND '01/07/10'
```

In SELECT use DD/MM/YY

- What is the amount of each product sale from '01/01/10' to '07/01/10'?

```
SELECT Barcode, ProductName, SUM(Amount) As Amount
FROM ProductSold
WHERE [Date] BETWEEN '01/01/10' AND '01/07/10'
GROUP BY Barcode, ProductName
```

- What is the amount of sale each day from '01/01/10' to '07/01/10'?

```
SELECT [Date], SUM(Amount) As Amount
FROM ProductSold
WHERE [Date] BETWEEN '01/01/10' AND '01/07/10'
GROUP BY [Date]
ORDER BY [Date]
```

2.4. Using AVG() Function

- AVG ()

- What is the Average score of each student in each semester?

StdID	SubID	Semester	Score
1	1	1	80
1	2	1	70
1	3	1	60
1	4	1	80
1	1	2	90
1	2	2	85
1	3	2	80
1	4	2	95

Table 4: Score

```
SELECT StdID, Semester,
AVG(Score) As Score
FROM Score
GROUP BY StdID, Semester
```

2.5. Using Min() and Max() Function

- Min() and Max()
 - What is the minimum score of student number 1 for all subject (Table 4)?

```
SELECT MIN(Score) as MinScore
FROM Student
WHERE StdID=1
```
 - What is the maximum score of student number 1 for all subject (Table 4)?

```
SELECT MAX(Score) AS MaxScore
FROM Student
WHERE StdID=1
```

2.6. Using HAVING

- Using HAVING to filter data as similar as to a WHERE clause. The difference between WHERE and HAVING is “WHERE filters data on specific rows and HAVING filters data on group of rows.”
- Using Aggregate Functions (SUM, COUNT, AVG, MIN, MAX) in the Condition Expressions with HAVING clause.

Using HAVING

- Example
 - In Table 3: Retrieve all products sold more than 5 from 01/01/10 to 07/01/10.

```
SELECT Barcode, SUM(Qty) As Qty
FROM ProductSold
WHERE [Date] BETWEEN '01/01/10' AND
      '01/07/10'
GROUP BY Barcode
HAVING SUM(Qty)>5
ORDER BY Barcode
```

3. JOINS BETWEEN TABLES

- Equijoins or Inner Joins (Joins of Equality)
- Natural Join
- Outer Join
 - Left Join
 - Right Join
- Self Join

Join between Tables

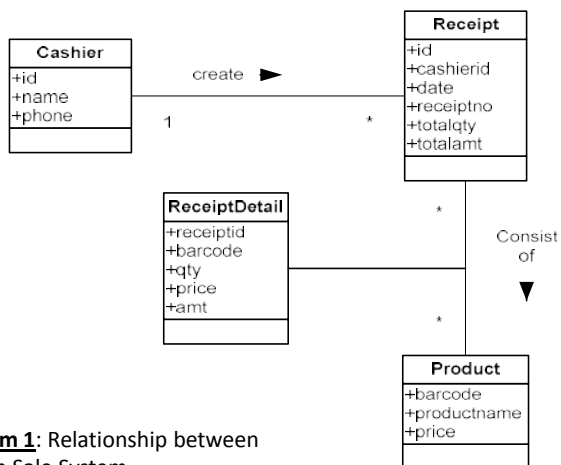


Diagram 1: Relationship between table in Sale System

3.1. Inner Join or Equijoins

- Equijoin (Inner Join) joins two tables with a common column in which each is usually the primary key.
- Syntax

```

SELECT T1.Column1, T2.Column2, ...
FROM Table1 T1, Table2 T2[, Table3 T3]
WHERE T1.COLUMN=T2.COLUMN
      [AND T1.COLUMN=T3.COLUMN]

(OR)

SELECT T1.Column1, T2.Column2, ...
FROM Table1 T1 INNER JOIN Table2 T2
      ON T1.Column=T2.Column [INNER JOIN Table3 T3
      ON T1.Column=T3.Column]
    
```

Inner Join or Equijoins

- Example

- In Diagram 1: Please use SQL to retrieve sale amount by each cashier name.

```
SELECT c.[name], SUM(a.totalamt) As saleamt
FROM Cashier c, Receipt a
WHERE c.[id]=a.cashierid
GROUP by c.[name]
```

(OR)

```
SELECT c.[name], SUM(a.totalamt) As saleamt
FROM Cashier c INNER JOIN Receipt a
ON c.[id]=a.cashierid
GROUP by c.[name]
```

3.2. Natural Join

- A Natural Join is nearly the same as the equijoins; however, the natural join differs from the equijoin by eliminating duplicate columns in the joining columns. The JOIN condition is the same, but the columns selected differ.
- Syntax

```
SELECT T1.*, T2.Column2 [, T3. Column3]
FROM Table1 T1, Table2 T2[, Table3 T3]
WHERE T1.Column=T2.Column
      [AND T1.Column=T3.Column]
```

Natural Join

- Example

- Retrieve all cashier and sale

```
SELECT a.*, b.totalqty, b.totalamt
FROM Cashier a, Receipt b
WHERE a.[id]=b.cashierid
```

(OR)

```
SELECT a.*, b.totalqty, b.totalamt
FROM Cashier a INNER JOIN Receipt b
ON a.[id]=b.cashierid
```

3.3. Outer Join

- An *outer join* is used to return all rows that exist in one table, even though corresponding rows do not exist in the joined table.
- Syntax

```
SELECT T1.Column1, T2.Column2
FROM Table1 T1 {LEFT|RIGHT}
OUTER JOIN Table2 T2
ON T1.Column=T2.Column
```

LEFT Outer Join

- Example

- Retrieve all products with its selling quantity

```
SELECT p.barcode, p.productname,  
       SUM(a.totalqty) As Qty  
FROM product p LEFT OUTER JOIN  
  ReceiptDetail a  
  ON p.barcode= a.barcode  
GROUP BY p.barcode, p.productname
```

RIGHT Outer Join

- Retrieve all receipt with cashier name (although receipt not join to cashier)

```
SELECT c.[name], a.receiptno,  
       a.[date], a.totalqty,  
       a.totalamt  
FROM cashier c RIGHT OUTER JOIN  
  Receipt a  
  ON c.[id]=a.cashierid  
ORDER BY a.[date] DESC
```

3.4. Self Join

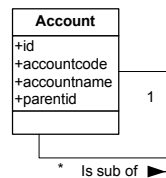
- The *self join* is used to join a table to itself, as if the table were two tables, temporarily renaming at least one table in the SQL statement using a table *alias*.
- Syntax

```
SELECT A.Column1, B.Column2, ...
FROM Table1 A, Table1 B
WHERE A.Column=B.Column
```

Self Join

- Example

```
SELECT A.accountcode,
       A.accountname
FROM Account A,
       Account B
WHERE A.parentid=b.[id]
```



4. Manipulating Data (INSERT, DELETE, UPDATE)

- INSERT Statement

- Syntax:

```
INSERT INTO table[(column1, column2, ...)]
VALUES ('value1', value2, ..., NULL,...)
```

- Example: Insert data to **Table 1**

```
INSERT INTO Student
VALUES (1, 'Mono', 'M', NULL, 'PP')
```

Not specify columns
mean all columns

(OR)

```
INSERT INTO Student(id, [name],
                    Gender, Phone, [Birth Place])
VALUES (1, 'Mono', 'M', NULL, 'PP')
```

Manipulating Data

- Inserting Data from Other Table

- Syntax

```
INSERT INTO table2(column1, column2[,...])
SELECT col1, col2[,...]
FROM table2 [WHERE condition] [group by ...]
```

- Example

```
INSERT INTO student([id], [name],
                    gender, [birth place])
SELECT [id], [name], gender, 'PP'
FROM StudentPP
```

Manipulating Data

- DELETE Statement

- Syntax

```
DELETE FROM Table  
[WHERE condition]
```

* Note: if you do not specify condition in delete clause then all data will be deleted.

- Example

```
DELETE FROM Student  
WHERE [birth place]='PP' AND phone='011'
```

Manipulating Data

- UPDATE Statement

- Syntax

```
UPDATE Table  
SET column1=value1  
    [, column2=value2]  
    [.....]  
[WHERE condition]
```

- Example

```
UPDATE Student  
SET [Birth Place]='Takeo'  
    , Phone='012'  
WHERE [id] BETWEEN 3 AND 8
```

5. Using Subquery

- A *subquery*, also known as a nested query, is a query embedded within the WHERE clause of another query to further restrict data returned by the query.
- Syntax

```
SELECT <columns list>
FROM table1
WHERE column1 {Operator|IN}
              (SELECT column1
               FROM Table2
               [WHERE condition]
               [GROUP BY <columns>]
               [HAVING condition exp])
```

* Note: Similarity in INSERT, DELETE and UPDATE Statement, we use subquery with WHERE clause.

Using Subquery

- Example
 - In diagram 1: Please write SQL Statement to retrieve all products which not yet exists in sale transactions.

```
SELECT barcode, productname
FROM product
WHERE barcode NOT IN
      (SELECT barcode
       FROM ReceiptDetail b)
```

Using Subquery

- Retrieve list of products that its sale quantity greater than 10 during 01/01/10 to 31/01/10 (Diagram 1).

```
SELECT barcode, productname
FROM Product
WHERE barcode IN (SELECT barcode
  FROM Receipt a, ReceiptDetail b
  WHERE a.[id]=b.receiptid AND
    (a.[date] BETWEEN '01/01/10' AND '10/31/10'))
GROUP BY barcode
HAVING SUM(qty)>10)
```

6. Combining Multiple Queries into One

- Using UNION Operator
 - Syntax

```
SELECT <columns list1>
FROM table1
[WHERE condition]
UNION [ALL]
SELECT <columns list2>
FROM table2
[WHERE condition]
```

If not specify **ALL** then result will eliminate Duplicated rows into one row only, otherwise all duplicated rows are resulted

Combining Multiple Queries into One

- Example
 - Suppose we have 3 tables (StudentPP, StudentTK, StudentKD) with the same columns. We want to query from these 3 tables into one result.

```
SELECT [id], [name], gender, address='PP'  
FROM StudentPP  
UNION ALL  
SELECT [id], [name], gender, address='TK'  
FROM StudentTK  
UNION ALL  
SELECT [id], [name], gender, address='KD'  
FROM StudentKD
```

Case Study with Mart POS System

- View sale by each date
- View sale by each product
- View sale by each month and year
- View the best sale to the poor sale of products
- View sale by each cashier

* *All the above views are defined in a specific duration.*