

Chapter 7: Introduction to SQL **SELECT** Statement

Modern Database Management
8th Edition

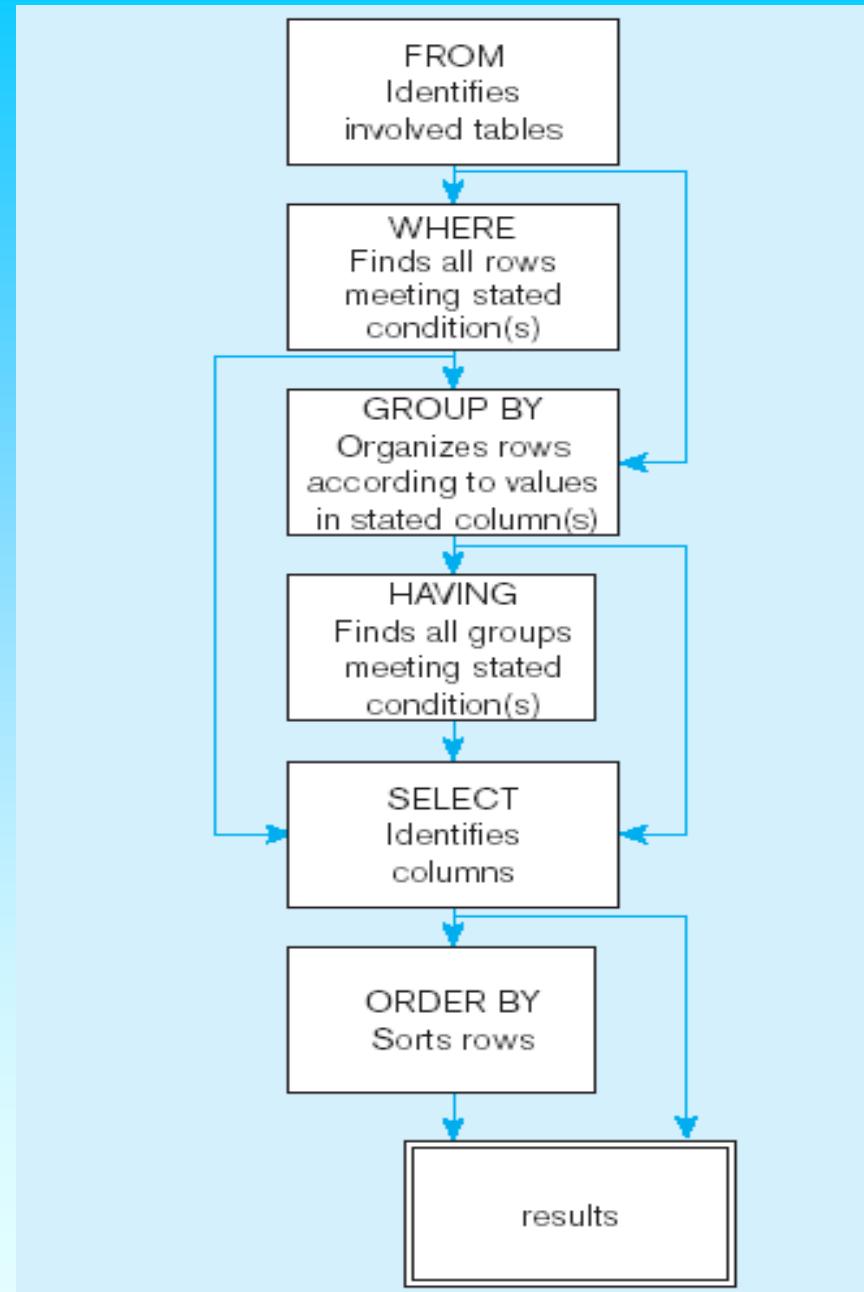
***Jeffrey A. Hoffer, Mary B. Prescott,
Fred R. McFadden***

SELECT Statement

- Used for queries on single or multiple tables
- Clauses of the SELECT statement:
 - **SELECT**
 - List the **columns** (and **expressions**) that should be returned from the query
 - **FROM**
 - Indicate the **table(s)** or **view(s)** from which data will be obtained
 - **WHERE**
 - Indicate the **conditions** under which a **row** will be **included** in the result
 - **GROUP BY**
 - Indicate **categorization of results**
 - **HAVING**
 - Indicate the **conditions** under which a **category (group)** will be **included**
 - **ORDER BY**
 - **Sorts the result** according to specified criteria

important

Figure 7-10 SQL statement processing order (adapted from van der Lans, p.100)



SELECT Example

- Find products with standard price less than \$275

```
SELECT PRODUCT_NAME, STANDARD_PRICE  
FROM PRODUCT_V  
WHERE STANDARD_PRICE < 275;
```

Table 7-3 Comparison Operators in SQL

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

Table 7-3: Comparison Operators in SQL

SELECT Example Using Alias

- Alias is an alternative column or table name

alias of customer column with
AS statement

```
SELECT CUST.CUSTOMER AS NAME,  
      CUST.CUSTOMER_ADDRESS  
  FROM CUSTOMER_V CUST  
 WHERE NAME = 'Home Furnishings';
```

execution starts from
FROM statement

SELECT Example Using a Function

- Using the COUNT *aggregate function* to find totals

```
SELECT COUNT(*) FROM ORDER_LINE_V  
WHERE ORDER_ID = 1004;
```

returns count of selected items

Note: with aggregate functions you can't have single-valued columns included in the SELECT clause

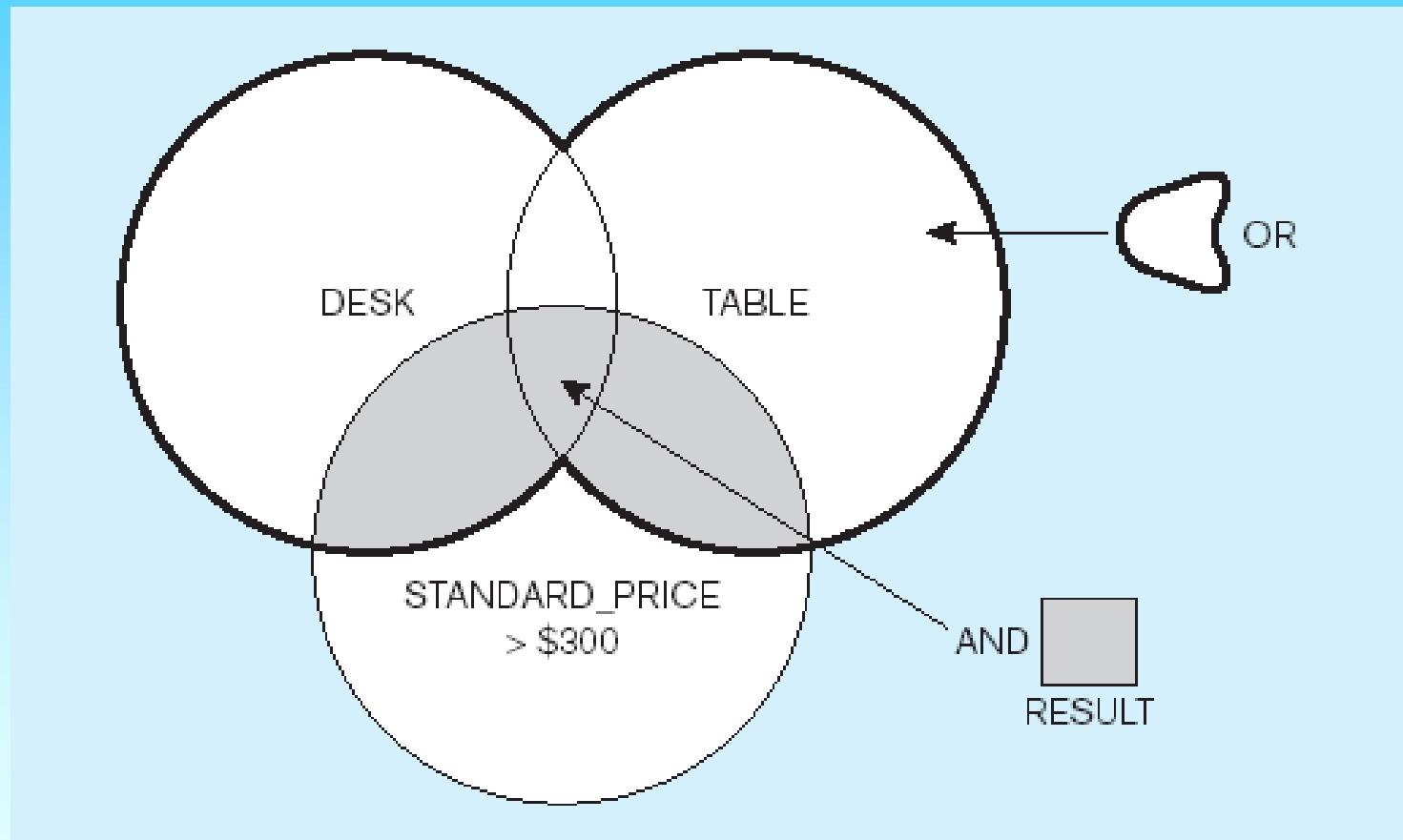
SELECT Example—Boolean Operators

- **AND, OR, and NOT** Operators for customizing conditions in **WHERE** clause

```
SELECT PRODUCT_DESCRIPTION, PRODUCT_FINISH,  
      STANDARD_PRICE    if we know character position we replace it with  
      FROM PRODUCT_V      underscores by the number of characters  
      WHERE (PRODUCT_DESCRIPTION LIKE '%Desk'  
             OR PRODUCT_DESCRIPTION LIKE '%Table')  
             AND UNIT_PRICE > 300;
```

Note: the **LIKE** operator allows you to compare strings using **wildcards**. For example, the % wildcard in '%Desk' indicates that all strings that have any number of characters preceding the word "Desk" will be allowed

Venn Diagram from Previous Query



SELECT Example – Sorting Results with the ORDER BY Clause

- Sort the results first by STATE, and within a state by CUSTOMER_NAME

```
SELECT CUSTOMER_NAME, CITY, STATE  
      FROM CUSTOMER_V  
     WHERE STATE IN ('FL', 'TX', 'CA', 'HI')  
ORDER BY STATE, CUSTOMER_NAME;
```

sort with first
then with
second

Note: the IN operator in this example allows you to include rows whose STATE value is either FL, TX, CA, or HI.

It is more efficient than separate OR conditions

SELECT Example—

Categorizing Results Using the GROUP BY Clause

- For use with aggregate functions

- Scalar aggregate:** single value returned from SQL query with aggregate function
 - Vector aggregate:** multiple values returned from SQL query with aggregate function (via GROUP BY)

```
SELECT CUSTOMER_STATE, COUNT(CUSTOMER_STATE)
  FROM CUSTOMER_V
 GROUP BY CUSTOMER_STATE;
```

Note: you can use single-value fields with aggregate functions if they are included in the GROUP BY clause

SELECT Example—

Qualifying Results by Categories Using the HAVING Clause

- For use with GROUP BY

```
SELECT CUSTOMER_STATE, COUNT(CUSTOMER_STATE)  
      FROM CUSTOMER_V  
    GROUP BY CUSTOMER_STATE  
HAVING COUNT(CUSTOMER_STATE) > 1;
```

Like a WHERE clause, but it operates on groups (categories), not on individual rows.

Here, only those groups with total numbers greater than 1 will be included in final result

SQL تعليمات SQL Statement

SELECT

الشكل العام لـ SELECT

```
SELECT      [DISTINCT] ColumnName(s)  
FROM      Table(s)  
[WHERE Condition]  
[GROUP BY ColumnName(s)]  
[HAVING Condition]  
[ORDER BY ColumnName(s)] ;
```

Tables

- COURSE (CRS#, CTITLE, C#HRS, C#HRSW)
- PREREQ (SUPCRS#, SUBCRS#, CRSCAT)
- OFFER (CRS#, OFF#, EMP#, LOCATION)
- TRAINER (EMP#, FNAME, LNAME, DOB, STATE, TEL#, JOB, SALARY, RAT)

اختيار أعمدة محددة من جدول
(*) اختيار كل أعمدة جدول

مثال ١ : عرض أرقام وأسماء جميع البرامج التدريبية المتاحة

```
SELECT CRS#, CTITLE  
FROM COURSE ;
```

مثال ٢ : عرض تفاصيل جميع البرامج التدريبية المتاحة

```
SELECT *  
FROM COURSE ;
```

- حذف الصفوف المتكررة DISTINCT

- مثال ٣ : عرض أسماء أماكن تنفيذ الدورات التدريبية مع حذف
الصفوف المتكررة من النتيجة

```
SELECT DISTINCT LOCATION  
FROM OFFER ;
```

الاسترجاع المشروط Conditional Retrieval عبارة WHERE

SELECT . . .
FROM . . .
WHERE [NOT] ColumnName
ComparisonOperator Literal

■ عوامل المقارنة Comparison Operators

= تساوي <> لا يساوي < أكبر من

<= أقل من أو يساوي > أكير من أو يساوي

الاسترجاع المشروط Conditional Retrieval عبارة WHERE

مثال ١ : أسماء الموظفين الذين يعملون في القاهرة

```
SELECT FNAME, LNAME  
FROM     TRAINER  
WHERE    STATE= 'Cairo';
```

مثال ٢ : أسماء الموظفين ومرتباتهم <= ٣٥٠٠٠

```
SELECT FNAME, LNAME, SALARY  
FROM     TRAINER  
WHERE    SALARY >= 35000 ;
```

القيم غير المعروفة NULLs

WHERE ColumnName IS [NOT] NULL

مثال ٣ : أرقام وأسماء الموظفين الذين لم يحدد لهم معدل أداء
RAT

```
SELECT EMP#, FNAME, LNAME  
FROM TRAINER  
WHERE RAT IS NULL;
```

ORDER BY ترتيب جدول النتائج

```
SELECT . . .
FROM ...           WHERE ...
ORDER BY ColumnName [ DESC ]
[,ColumnName [ DESC ] ...];
```

مثال ١: اسم العائلة والاسم الأول للموظفين الذين يعملون في القاهرة مرتبة تصاعدياً باسم العائلة

```
SELECT LNAME, FNAME
FROM TRAINER
WHERE STATE = 'Cairo'
ORDER BY LNAME;
```

ترتيب جدول النتائج بالرقم النسبي لعمود

ترتيب جدول النتائج بالرقم النسبي لحقل بيان (عمود)

```
SELECT ColumnName, columnName, ...
FROM ...
WHERE ...
ORDER BY columnNbr [ DESC ]
[,columnNbr [ DESC ] ...];
```

مثال ٢: اسم العائلة والعمل والمرتب للموظفين الذين يعملون في الجيزة
- رتب الجدول تنازلياً بالمرتب وتصاعدياً بالعمل

```
SELECT LNAME, JOB, SALARY
FROM TRAINER
WHERE STATE = 'Giza'
ORDER BY 3 DESC, 2; numbers of the selected tables
```

تعدد الشروط Multiple Conditions

ربط شروط البحث AND & OR

مثال ١: أرقام وأسماء الموظفين الذين يعملون كمبرمجين ومرتباتهم أقل من ٣٢٠٠٠

```
SELECT EMP#, FNAME, JOB, SALARY  
FROM TRAINER  
WHERE JOB = 'PRG' AND SALARY < 32000;
```

مثال ٢: أرقام وأسماء الموظفين الذين يعملون كمبرمجين أو مرتباتهم أقل من ٣٢٠٠٠

```
SELECT EMP#, FNAME, JOB, SALARY  
FROM TRAINER  
WHERE JOB = 'PRG' OR SALARY < 32000;
```

تابع تعدد الشروط ربط شروط البحث AND & OR

مثال ٣ : الدورات التي حصل عليها المتدربان ١٦ و ١٧ و سُجّلت تقديراتها

```
SELECT *
FROM TRAINEE
WHERE EMP# = 16 OR EMP# = 17
AND GRADE IS NOT NULL;
```

■ يجب استخدام الأقواس

```
SELECT *
FROM TRAINEE
WHERE (EMP# = 016 OR EMP# = 017)
AND GRADE IS NOT NULL;
```

تابع تعدد الشروط Multiple Conditions نفي الشروط Negation Of Conditions

مثال ٤: تفاصيل البرامج المطلوبة لجميع البرامج التدريبية عدا
البرمجين ٧٠٤ و ٧٠٦

```
SELECT *  
FROM PREREQ  
WHERE NOT (SUPCRS# = 704  
           OR SUPCRS# = 706);
```

BETWEEN, IN, LIKE العوامل العلاقة

اختيار صفوف بقيم في مدى محدد
BETWEEN ... AND

SELECT ...

FROM ...

**WHERE ColumnName [NOT] BETWEEN
LowValue AND HighValue;**

مثال ١ : أرقام الموظفين وأسماؤهم الأولى التي تبدأ بالحروف

A, B, C, D, E, F

try this example with in

SELECT EMP#, FNAME

FROM TRAINER

WHERE FNAME BETWEEN 'A%' AND 'F%';

العوامل العلاقية BETWEEN, IN, LIKE

IN اختيار صفوف بقيم محددة بقائمة

SELECT ... FROM ...

WHERE

ColumnName [NOT] IN (List-of-values) ;

مثال ٢: الدورات التي يقوم بتنفيذها المدربان ١ و ١٠

SELECT *

FROM TRAINER

WHERE EMP# IN (1, 10) ;

العوامل العلاقية BETWEEN, IN, LIKE

اختيار صيغ طبقاً لتركيبة حروف LIKE

SELECT ... FROM ...

WHERE

ColumnName [NOT] LIKE (Searching-String)

مثال ٣: أسماء الموظفين الأولى التي بها الحروف AH في الموقع الثاني والثالث من الاسم

```
SELECT FNAME, LNAME  
FROM TRAINER  
WHERE FNAME LIKE '_AH%';
```

القيمة المحسوبة Computed Values العبارات الحسابية في SELECT

```
SELECT ColumnName,  
       Expression, ColumnName, ...
```

```
FROM ... WHERE ...
```

```
ORDER BY ColumnNbr ;
```

مثال ١ : تقرير بأرقام وأسماء البرامج ومدة كل برنامج بالأسبوع مرتبًا بعدد الأسابيع

```
expression  
SELECT CRS#, CTITLE, C#HRS/C#HRSW  
FROM COURSE  
ORDER BY 3 ;
```

القيمة المحسوبة العبارات الحسابية في WHERE

```
SELECT ColumnName, Expression, ColumnName,...  
FROM ...  
WHERE ... ORDER BY ColumnNbr ;
```

مثال ٢ : تقرير بأرقام وأسماء البرامج ومدة كل منها بالأسبوع مرتبًا بعدد الأسابيع للبرامج التي

مدتها أكثر من ٤ أسابيع ووضع عنوان لعدد الأسابيع “

```
SELECT CTITLE, '# OF WEEKS =', C#HRS/C#HRSTW  
FROM COURSE  
WHERE C#HRS/C#HRSTW > 4  
ORDER BY 3 ;
```

دوال الأعمدة Column Functions

القيمة المتوسطة لقيم عمود AVG

```
SELECT AVG ([ DISTINCT ] ColumnName)  
FROM ...  
WHERE ... ;
```

مثال ١ : القيمة المتوسطة لمعدل الموظفين العاملين بالقاهرة

```
SELECT AVG (RAT) average values  
FROM TRAINER  
WHERE STATE = 'Cairo' ;
```

دوال الأعمدة Column Functions

مجموع قيم عمود SUM

```
SELECT SUM ([ DISTINCT ] ColumnName)  
FROM ...  
WHERE ... ;
```

مثال ٢ : مجموع مرتبات أخصائيي ومبرمجي قواعد البيانات بالقاهرة

```
SELECT SUM (SALARY)  
FROM TRAINER  
WHERE (JOB = 'DBA' OR JOB = 'DBP')  
AND STATE = 'Cairo' ;
```

دوال الأعمدة Column Functions

وأكبر قيمة في قيم عمود MIN أقل قيمة في قيم عمود MAX

```
SELECT MIN ([ DISTINCT ] ColumnName)
```

```
FROM ...
```

```
WHERE ... ;
```

```
SELECT MAX ([ DISTINCT ] ColumnName)
```

```
FROM ...
```

```
WHERE ... ;
```

مثال ٣ : تاريخ ميلاد أكبر وأصغر مبرمج

```
SELECT MAX(DOB), MIN(DOB)
```

```
FROM TRAINER
```

```
WHERE JOB = 'PRG' ;
```

دوال الأعمدة Column Functions

عدد صفوف جدول النتائج COUNT

```
SELECT COUNT(*)  
FROM ...  
WHERE ... ;
```

مثال ٤ : عدد موظفي ”الجيزة“ ومتوسط مرتباتهم

```
SELECT COUNT(*), AVG(SALARY)  
FROM TRAINER  
WHERE STATE = 'Giza' ;
```

دوال الأعمدة Column Functions

عدد صفوف جدول النتائج بدون القيم المتكررة أو NULLs

```
SELECT COUNT(DISTINCT ColumnName)  
FROM ...  
WHERE ... ;
```

مثال ٥ : عدد الأعمال غير المتكررة التي يقوم بها الموظفون

```
SELECT COUNT(DISTINCT JOB)  
FROM TRAINER;
```

المجموعات Grouping

تجميع الصفوف في مجموعات فرعية GROUP BY

```
SELECT  
    ColumnName[,ColumnName][,ColumnFunction], ...  
FROM ...  
WHERE ...  
GROUP BY ColumnName[,ColumnName ...]  
ORDER BY ColumnName [DESC][,ColumnName  
[DESC]...]  
SequenceNumber [DESC]  
,SequenceNumber [DESC]...];
```

المجموعات Grouping

تجميع الصفوف في مجموعات فرعية GROUP BY

مثال ١ : ما مجموع مرتبات الموظفين بكل موقع عمل Location عدا مرتبات المبرمجين؟ رتب النتيجة طبقاً للمجموع

```
SELECT STATE, SUM (SALARY)
FROM TRAINER
WHERE NOT JOB = 'PRG'
GROUP BY STATE
ORDER BY 2 ;
```

المجموعات Grouping

ترشيح المجموعات الفرعية GROUP BY ... HAVING

مثال ٢ : متوسط مرتبات كل عمل JOB عدا DBA بكل موقع عمل Location للأعمال التي يقوم بها أكثر من موظف. رتب النتيجة تنازليا بمتوسط المرتبات تصاعديا داخل العمل داخل موقع العمل.

```
SELECT STATE, 'JOB NAME IS', JOB,  
       AVG (SALARY)  
FROM TRAINER  
WHERE JOB <> 'DBA'  
GROUP BY STATE, JOB  
HAVING COUNT (*) > 1  
ORDER BY STATE, JOB, 4 DESC ;
```

المجموعات Grouping

ترشيح المجموعات الفرعية GROUP BY ... HAVING

سلسل تنفيذ SELECT

- اختيار صفات الجدول التي تتحقق شروط WHERE
- تكون GROUP BY مجموعات فرعية من الصفات المختارة
- حذف المجموعات الفرعية التي لا يتواجد فيها شروط HAVING
- تنفيذ دوال العمود أو التعبيرات الحسابية في SELECT على المجموعات الفرعية الناتجة في الخطوة السابقة
- ترتيب ناتج الخطوة السابقة طبقاً لعبارة ORDER BY.

ask from here

Using and Defining Views

- Views provide users controlled access to tables
- Base Table –table containing the raw data
- Dynamic View
 - A “virtual table” created dynamically upon request by a user
 - No data actually stored; instead data from base table made available to user
 - Based on SQL SELECT statement on base tables or other views
- Materialized View
 - Copy or replication of data
 - Data actually stored
 - Must be refreshed periodically to match the corresponding base tables

Sample CREATE VIEW

```
CREATE VIEW EXPENSIVE_STUFF_V AS  
SELECT PRODUCT_ID, PRODUCT_NAME, UNIT_PRICE  
FROM PRODUCT_T  
WHERE UNIT_PRICE >300  
WITH CHECK_OPTION;
```

- View has a name
- View is based on a **SELECT** statement
- **CHECK_OPTION** works **only** for
 updateable views and prevents updates
 that would create rows not included in
 the view

Advantages of Views

- Simplify query commands
- Assist with data security (but don't rely on views for security, there are more important security measures)
- Enhance programming productivity
- Contain most current base table data
- Use **little storage space**
- Provide **customized view** for user
- Establish physical data independence

Disadvantages of Views

- Use processing time each time view is referenced
- May or may not be directly updateable

■ Creating indexes

- DBMS uses Indexes to **Speed up** random/sequential access to base table data
- Although users do not directly refer to indexes when writing any SQL command, the **DBMS recognizes** which **existing indexes** would improve query performance
- Indexes are usually created for both **primary** and **foreign keys** and both **single** and **compound keys**
- Indexes could be in **ascending** or **descending** sequence
- Example
 - **CREATE INDEX NAME_IDX ON CUSTOMER_T(CUSTOMER_NAME)**
 - This makes an index for the CUSTOMER_NAME field of the CUSTOMER_T table
- To **remove** the index
 - **Drop INDEX NAME_IDX**