**Data Definition Language**

**&**

**Data Manipulation Language**

# Database Languages and Interfaces

Because a database supports a number of user groups, the DBMS must have languages and interfaces that support each user group.

## DBMS Languages

1. **Data Definition Language (DDL)**

DDL – the **data definition language**, used by the DBA and database designers to define the conceptual and internal schemas.The DBMS has a DDL compiler to process DDL statements in order to identify the schema constructs, and to store the description in the catalogue.In databases where there is a separation between the conceptual and internal schemas, DDL is used to specify the conceptual schema, and SDL, **storage definition language**, is used to specify the internal schema.

For a true three-schema architecture, VDL, **view definition language**, is used to specify the user views and their mappings to the conceptual schema. But in most DBMSs, the DDL is used to specify both the conceptual schema and the external schemas.

**2. Data Manipulation Language**

Once the schemas are compiled, and the database is populated with data, users need to manipulate the database. Manipulations include retrieval, insertion, deletion and modification.The DBMS provides operations using the DML, **data manipulation language**.

**3. SQL(Structured Query languages)**

In most DBMSs, the VDL, DML and the DML are not considered separate languages, but a comprehensive integrated language for conceptual schema definition, view definition and data manipulation. Storage definition is kept separate to fine-tune the performance, usually done by the DBA staff.

An example of **a comprehensive language: SQL**, which represents a VDL, DDL, DML as well as statements for constraint specification, etc.

* An example of a comprehensive language: SQL, which represents a VDL, DDL, DML as well as statements for constraint specification, etc.

**SQL**

* SQL is an ANSI (American National Standards Institute) standard.
* SQL is a database computer language designed for managing data in relational database management systems (RDBMS).
* SQL is a standard language for accessing and manipulating databases.
* SQL was developed at IBM by Donald D. Chamberlin and Raymond F. Boyce in the early 1970s.
* Often pronounced like "sequel“.
* SQL can create new databases, new tables in a database, execute queries against a database, retrieve data from a database, insert records in a database, update records in a database, delete records from a database, create stored procedures in a database, create views in a database, set permissions on tables, procedures, and views
* When a user wants to get some information from a database file, he can issue a ***query***. A query is a user–request to retrieve data or information with a certain condition.
* SQL is a query language that allows user to specify the conditions. (instead of algorithms) The user specifies a certain condition.The program will go through all the records in the database file and select those records that satisfy the condition.(searching).
* Statistical information of the data.
* The result of the query will then be stored in form of a table.

**Categories of SQL statements**

* + Data Definition Language (DDL) Statements
  + Data Manipulation Language (DML) Statements
  + Transaction Control Statements
  + Session Control Statements
  + System Control Statement
  + Embedded SQL Statements

**Data Definition Language (DDL) Statements**

DDL statements are used to build and modify the structure of your tables and other objects in the database. When you execute a DDL statement, it takes effect immediately.

Some examples:

* CREATE - to create objects in the database
* ALTER - alters the structure of the database
* DROP - delete objects from the database
* RENAME - rename an object

**DDL Statements for Database, Table, View,Constraints**

1. **Database**

**CREATE DATABASE**

Installing a database management system (DBMS) on a computer allows you to create and manage many independent databases.The CREATE command can be used to establish each of these databases on your platform.

**Syntax**:CREATE DATABASE database\_name

**Example**:CREATE DATABASE db\_students

**DROP DATABASE**

Removes a database from the system.

**Syntax**:DROP DATABASE database\_name

**Example:** DROP DATABASE db\_students

**SHOW DATABASE**

The *show databases* SQL command is used to list all databases.

**Syntax**:show databases;

1. **User**

The Data Control Language (DCL) authorizes users and groups of users to access and manipulate data. Main DDL Statements for Database USER

* Create USER

Syntax: CREATE USER username SET PASSWORD 'password'

Example: CREATE USER harry SET PASSWORD 'cat‘

Syntax: ALTER USER username SET PASSWORD 'password'

Example: ALTER USER harry SET PASSWORD 'cat‘

* Drop USER

Syntax: DROP USER username

Example: DROP USER harry

* Grant /Revoke USER
  + GRANT authorizes one or more users to perform an operation or a set of operations on an object.

Syntax:

GRANT SELECT, UPDATE ON My\_table TO some\_user, another\_user;

* + REVOKE eliminates a grant, which may be the default grant.

Syntax:

Revoke SELECT, UPDATE ON My\_table TO some\_user, another\_user;

1. **Table**
   1. **Data Types**

A **data type** (or **datatype**) is a classification identifying one of various types of data, such as floating-point, integer, or Boolean, that determines the possible values for that type; the operations that can be done on that type; and the way the values of that type are stored.Every column in a table must be allocated a data type.

SQL supports a wide range of different data types.

**Boolean**

Boolean values are true/false types of data. A Boolean table column will contain either string values of “True” and “False” or the numeric equivalent representation, with 0 being false or 1 being true.

* + ("TRUE" / "FALSE")
  + ( 1 / 0 )

**Numbers**

A range of numeric data types are provided to support integers and decimal numbers.

INTEGER: This data type is useful for general number columns, supporting negative and positive numbers.

FLOAT: This more general numeric data type supports decimal places.

**CHAR**

This is the basic character based datatype.The length of the CHAR attribute is denoted in brackets following the CHAR key word. For example, a column requiring four characters would be defined CHAR(4).If no length is provided, then the attribute defaults to a single character length. The CHAR datatype is fixed length - this means that if the value 'S1' is stored in an attribute specified as CHAR(4), then the DBMS will pad out the remaining characters with spaces, i.e. the whole four characters will always be used. FLOAT(4,2) allows for two digits after the decimal point (e.g. 12.45).

**VARCHAR:**

This is a more flexible character data type. VARCHAR is a variable length type; the maximum number of characters is again denoted in brackets following the keyword - for example, VARCHAR(30) would provide an attribute with a maximum of 30 characters. However, if fewer than 30 characters are used, the DBMS does not pad out the remaining characters. The VARCHAR datatype can therefore make significant savings on space.

**Binary**

Binary data types allow you to store any type of binary data, including entire files of up to 2GB. Data types in the binary category include:

* + binary(n) variables store n bytes of fixed-size binary data. They may store a maximum of 8,000 bytes.
  + varbinary(n) variables store variable-length binary data of approximately n bytes. They may store a maximum of 8,000 bytes.
  + image variables store up to 2 gigabytes of data and are commonly used to store any type of data file(not just images).

**DateTime**

A number of specific data types are provided to support date and time data items. The following are examples

**DATE:** This is useful for storing values which are numeric in nature, but that we know are of type date. An attribute of type DATE has 10 positions - these are split to represent the YEAR, MONTH and DAY components, typically in the form YYYY-MM-DD.

The DATE type supports a wide variety of formatting options, which can be very helpful for data output purposes. There are also built in functions that allow manipulation of attributes of DATE type - examples are ADD\_MONTHS, MONTHS\_BETWEEN, NEXT\_DAY. You should consult an SQL reference to check the full list of functions supported.

**TIME:** An attribute of type TIME has 8 positions which are split to represent the HOUR, MINUTE and SECOND components, typically in the form HH:MM:SS. As with DATE, the TIME data type provides additional formatting options that can be used to output the time data in alternative formats.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Data Types** | **Naming in different database platforms** | | | | |
|  | **Access** | **SQL-Server** | **Oracle** | **MySQL** | **PostgreSQL** |
| *Boolean* | Yes/No | Bit | Byte | N/A | Boolean |
| *integer* | Number (integer) | Int | Number | Int Integer (synonyms) | Integer Int |
| *float* | Number (single) | Float Real | Number | Float | Numeric |
| *string (fixed)* | N/A | Char | Char | Char | Char |
| *string (variable)* | Text | Varchar | Varchar Varchar2 | Varchar | Varchar |
| *binary object* | OLE Object Memo | Binary (fixed up to 8K) Varbinary (<8K) Image (<2GB) | Long Raw | Blob Text | Binary Varbinary |

**DDL Statements For Tables**

* + **CREATE TABLE** -
    - The CREATE TABLE statement is used to create a table in a database.

Syntax: **CREATE TABLE table\_name  
(  
column\_name1 data\_type,  
column\_name2 data\_type,  
column\_name3 data\_type,  
....  
)**

Example: CREATE TABLE Persons  
(  
id int,name varchar(40),FirstName varchar(255),  
Address varchar(255),City varchar(255)  
**}**

* + **ALTER TABLE** –
    - ALTER is used to add / remove / modify the columns of a table.
    - Syntax:

**ALTER TABLE table\_name ADD [COLUMN] column\_declare**

**ALTER TABLE table\_name DROP [COLUMN] column\_name**

**ALTER TABLE table\_name ALTER [COLUMN] column\_name SET default\_expr**

**ALTER TABLE table\_name ALTER [COLUMN] column\_name DROP DEFAULT**

The ADD [COLUMN] form adds a new column definition to the table.The DROP [COLUMN] form drops the column with the name from the table. ALTER [COLUMN] column\_name SET default\_expr alters the default value for the column.ALTER [COLUMN] column\_name DROP DEFAULT removes the default value set for the column.

* + **DROP TABLE** –

Removes the table(s) from the database.

**Syntax**: **DROP TABLE [ IF EXISTS ] table\_name1, table\_name2, ....**

**Example:DROP TABLE PERSON;**

The IF EXISTS clause will drop the table only if it exists. If this clause is not present an error is generated if the table does not exist.

* + **SHOW TABLES**

Itreturns a list of tables in the database.

**Syntax: select \* from information\_schema.tables;**

* + **DESCRIBE table\_name** :
    - It describe the details of a table. This command provides information about the columns of the table. It shows the column names, the type / size and scale (if applicable) and other useful information.
    - Syntax: **DESC table\_name**
  + **RENAME TABLE old\_Tablename TO new\_Tablename**
    - **Syntax:** exec sp\_rename oldtable\_name,newtable\_name
    - **Example**:exec sp\_rename Persons, Persons\_new

**View**

View is a virtual table. A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database. You can add SQL functions, WHERE, and JOIN statements to a view and present the data as if the data were coming from one single table.

* **CREATE VIEW**

**Syntax:**

CREATE VIEW view\_name AS SELECT column\_name(s) FROM table\_name WHERE condition

**Example:**

CREATE VIEW ViewOfTableA AS SELECT col1\_id FROM TableA

* **Alter View**

The Alter View modify the existing created view.The ALTER VIEW  statement is used to modify the created view enlisted all the list of field selected in select query.

**Syntax:**

ALTER VIEW view\_name AS SELECT column\_name(s) FROM table\_name  
 WHERE condition

**Example:**

ALTER VIEW ViewOfTableA AS SELECT col1\_id FROM TableA Where col1\_id>5

* **DROP VIEW**

Removes a view from the database. A view can be changed by dropping and recreating it.

**Syntax: DROP VIEW table\_name**

**Example:** DROP VIEW ViewOfTableA;

1. **Constraints**

Constraints are used to limit the type of data that can go into a table.Constraints can be specified when a table is created (with the CREATE TABLE statement) or after the table is created (with the ALTER TABLE statement).Common Constraints are Common Constraints NOT NULL,UNIQUE,PRIMARY KEY,FOREIGN KEY,CHECK,DEFAULT

* **Adding constraints**

ALTER TABLE ADD CONSTRAINT t adds a table-level constraint to an existing table.

**Syntax:** ALTER TABLE table\_name{ ADD constraint\_clause,}

* **Dropping Constraints**

**Syntax:**

ALTER TABLE table\_name

{

DROP {PRIMARY KEY | FOREIGN KEY constraint-name | UNIQUE constraint-name | CHECK constraint-name | CONSTRAINT constraint-name}

}

Data Manipulation Language **(DML)**

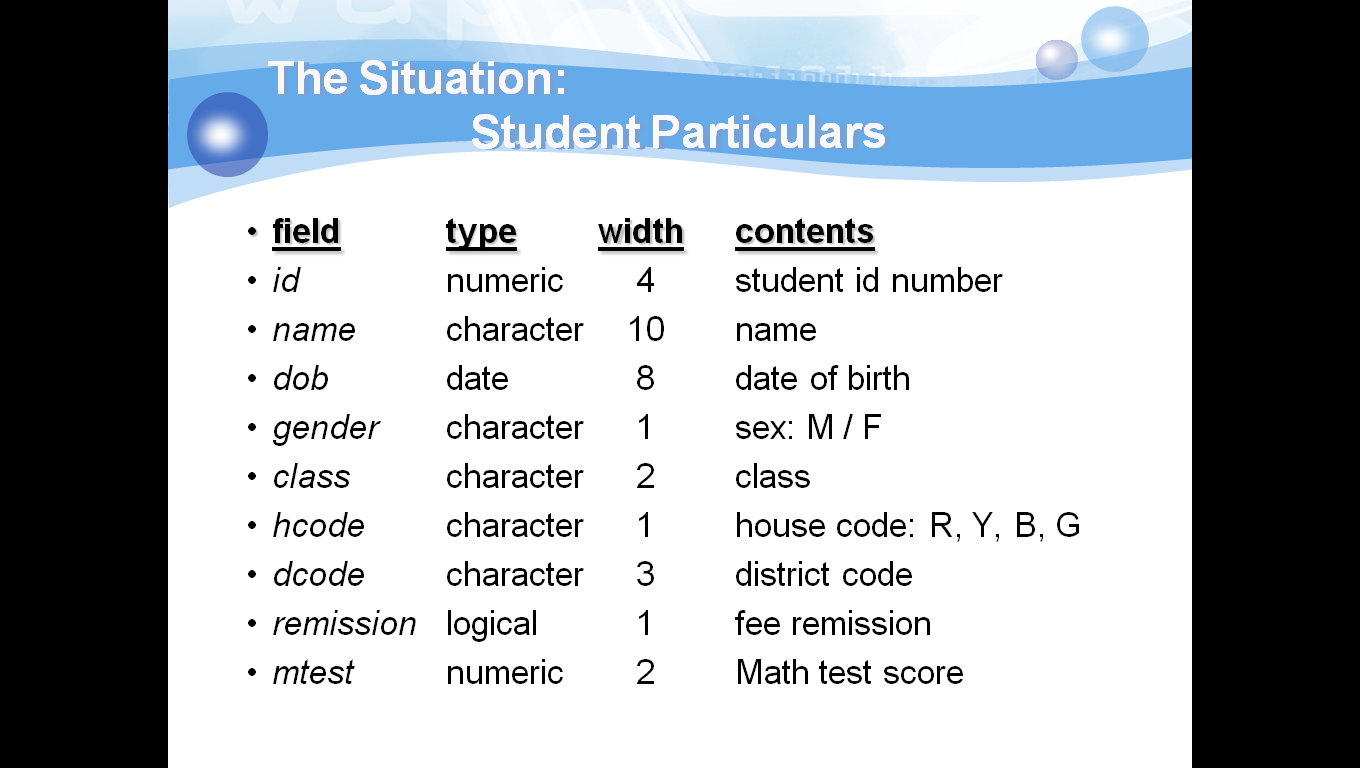
SQL (Structured Query Language) is syntax for executing queries. But the SQL language also includes syntax to update, insert, and delete records. These query and update commands together form the Data Manipulation Language (DML) part of SQL. Data Manipulation Language comprises the 'SQL-data change' statements [[2]](http://en.wikipedia.org/wiki/Data_Manipulation_Language#cite_note-SQL92.2C_statements_by_function-1), which modify stored data but not the schema or database objects. Different levels of user access can be controlled by the Database Administrator (DBA) - for example certain users may be restricted to using SELECT statements only, thus prohibiting them from altering the data stored in the database via the INSERT, UPDATE or DELETE statements. Data Manipulation is:

* Insertion of new information into the database.
* Modification of information in the database.
* Deletion of information in the database.
* Retrieval of information from the database.

The goal of DML is to provide efficient human interaction with the system. There are two types of DML:

* **Procedural:** the user specifies what data is needed and how to get it. A low-level or procedural DML allows the user to specify what data is needed and how to obtain it. This type of DML typically retrieves individual records from the database and processes each separately. In this language, the looping, branching etc. statements are used to retrieve and process each record from a set of records.
* **Nonprocedural:** the user only specifies what data is needed. It is Easier for user and may not generate code as efficient as that produced by procedural languages. A high-level or non-procedural DML allows the user to specify what data is required without specifying how it is to be obtained. Many DBMSs allow high-level DML statements either to be entered interactively from a terminal or to be embedded in a general-purpose programming language. The end-users use a high-level query language to specify their requests to DBMS to retrieve data. Usually a single statement is given to the DBMS to retrieve or update multiple records. The DBMS translates a DML statement into a procedure that manipulates the set of records.

**Consider The Student Particulars Below**

****

**The INSERT INTO Statement**

This command is used to add record(s) to a table. While inserting a record using insert statement, the number of records being entered should match the columns of the table. In case the number of items being entered are less than the number of columns, in that case the field names also need to be specified along with insert statement.

Syntax: INSERT INTO table\_name VALUES (value1, value2,....)

Example: INSERT INTO student VALUES (9801, ‘Peter’,’1986-04-06’ ,‘M’,’1A’,’R’,’ssp’, ‘F’,72);

You can also specify the columns for which you want to insert data:

Syntax: INSERT INTO table\_name (column1, column2,...) VALUES (value1, value2,....)

Example: INSERT INTO student (name, gender, mtest ) VALUES (‘Manila’, ’F’, 50);

You can also insert record from another table

Syntax: INSERT INTO table1 SELECT \* FROM table2 WHERE condition

Example: INSERT INTO student SELECT \* FROM student\_backlog WHERE class=‘1B’

**The UPDATE Statement**

The UPDATE statement is used to modify the data in a table.Once data has been inserted into a table in the database, we can access individual attributes to update them.The data may require updating either because of some change in the data value we now need to store, or as the result of some mistake having been discovered, such as the misspelling of a person's name.

Syntax:

UPDATE table\_name

SET column\_name = new\_value

WHERE column\_name = some\_value

Note: Notice the WHERE clause in the UPDATE syntax. The WHERE clause specifies which record or records that should be updated. If you omit the WHERE clause, all records will be updated!

Example:

Consider a situation where we wish to update the maths test score , such as changing the mtest field to 25 of student ‘Peter’

UPDATE student set mtest=75 WHERE name=‘Peter’

Update several columns in a row

UPDATE student set mtest=25,remission=‘T’ WHERE name=‘Peter’

**The DELETE Statement**

The DELETE FROM operation can be performed on individual or multiple rows of data in a table. The WHERE clause can be used to identify which rows are to be removed. Note, however, this does not remove the actual table from the database - only the data content of the table.

Syntax: DELETE FROM table\_name WHERE column\_name = some\_value

Example:

The SQL statement shown below deletes the row in the students table where the name attribute matches the value ‘Jenny’.

DELETE FROM student WHERE name=‘Jenny’

It is possible to delete all rows in a table without deleting the table. This means that the table structure, attributes, and indexes will be intact:

DELETE FROM table\_name Or

DELETE \* FROM table\_name

**The SELECT Statement**

The SELECT statement is used to select data from a database.The result is stored in a result table, called the result-set. The statement begins with the SELECT keyword. The basic SELECT statement has 3 clauses:

* SELECT
* FROM
* WHERE

The SELECT clause specifies the table columns that are retrieved. The FROM clause specifies the tables accessed. The WHERE clause specifies which table rows are used. The WHERE clause is optional; if missing, all table rows are used.

Syntax:

SELECT [ALL / DISTINCT] expr1 [AS col1], expr2 [AS col2]

FROM tablename WHERE condition

The query will select rows from the source tablename and output the result in table form. Expressions expr1, expr2 can be :

* + - a column, or
    - an expression of functions and fields.

And col1, col2 are their corresponding column names in the output table. DISTINCT will eliminate duplication in the output while ALL will keep all duplicated rows.

condition can be :

* an inequality, or
* a string comparison(using logical operators AND, OR, NOT.)

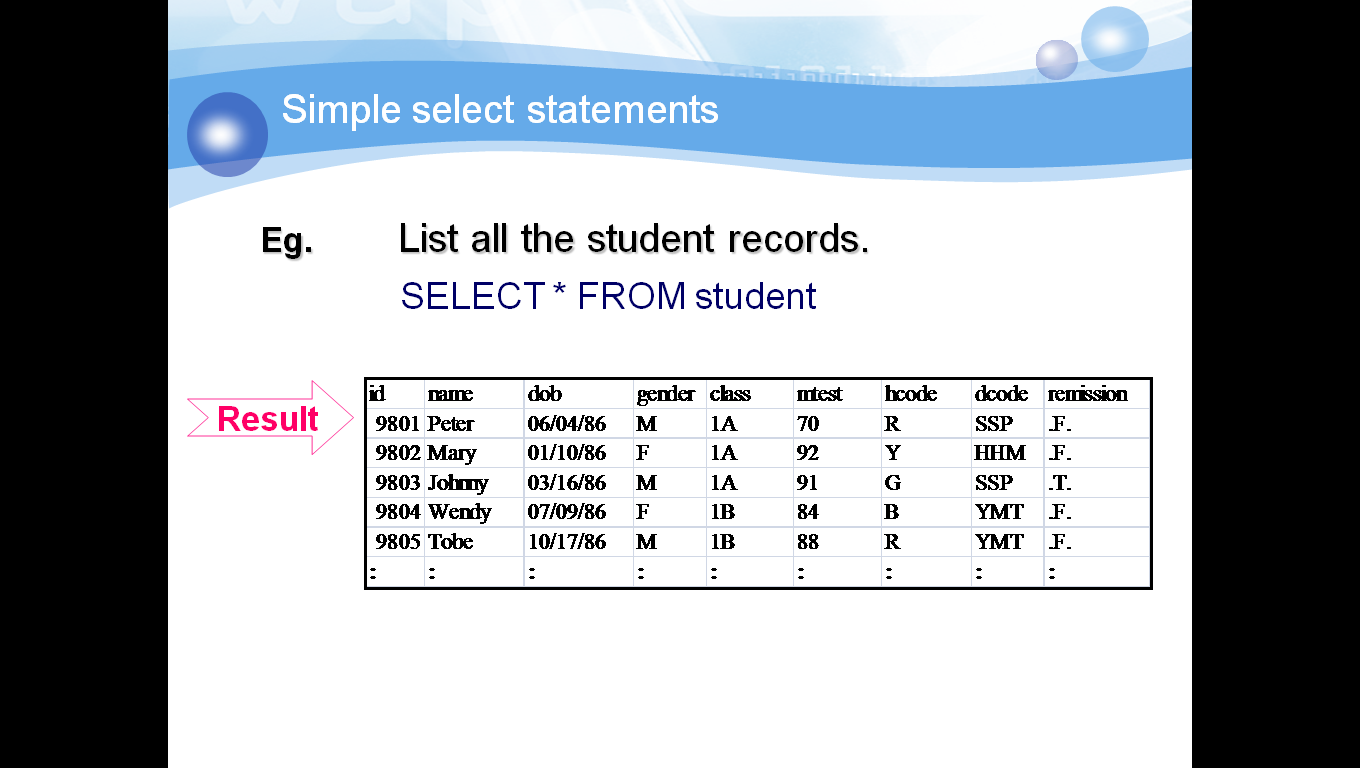
**Basic structure of an SQL query**

|  |  |
| --- | --- |
| General Structure | SELECT, ALL / DISTINCT, \*,  AS, FROM, WHERE |
| Comparison | IN, BETWEEN, LIKE "% \_" |
| Grouping | GROUP BY, HAVING,  COUNT( ), SUM( ), AVG( ), MAX( ), MIN( ) |
| Display Order | ORDER BY, ASC / DESC |
| Logical Operators | AND, OR, NOT |
| Output | INTO TABLE / CURSOR  TO FILE [ADDITIVE], TO PRINTER, TO SCREEN |
| Union | UNION |

Example: List all the student records.

SELECT \* FROM student

Result:



It may be that we are only interested in a particular number of attributes in the relation. In such cases we can specify the columns in the select statements

Example: SELECT name, hcode, class FROM student

**The DISTINCT Keyword**

We can limit the query to remove duplicates. In a table, some of the columns may contain duplicate values. This is not a problem; however, sometimes you will want to list only the different (distinct) values in a table. This can be achieved by the use of the keyword UNIQUE or DISTINCT being added after the SELECT keyword.

Example: SELECT DISTINCT dcode FROM student

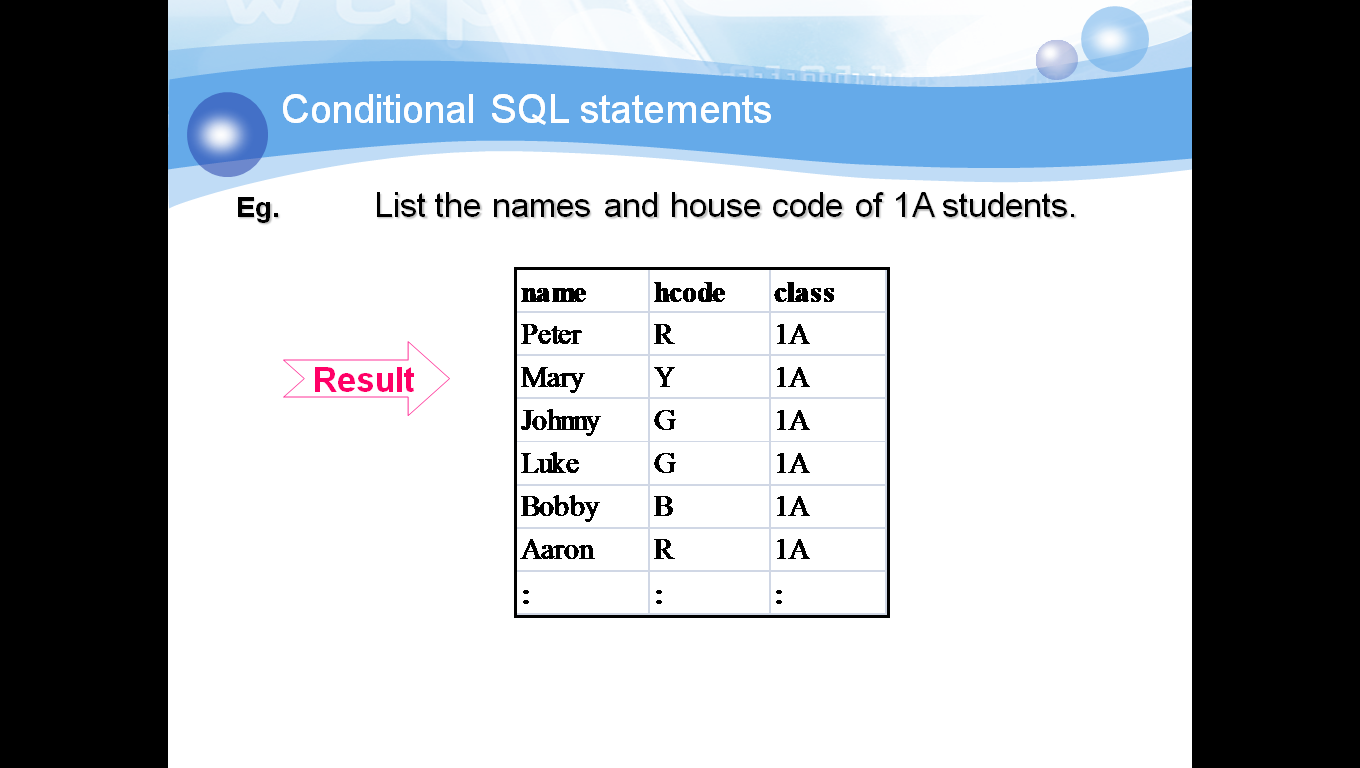
**The WHERE Clause**

The WHERE clause is used to filter records. We can put conditions on the SQL statement such that only records with particular attribute values will be returned from the database. The WHERE clause sets a conditional statement, and it can be used with any type of SQL query. As the select query executes, SQL processes one row at a time. Each time the conditional statement is met (returns true), a row is returned as a result. SQL WHERE is essentially, a filtering mechanism for SQL queries and is a tremendous asset to any aspiring SQL developer. In the example below, we are using the WHERE clause to filter out rows and only selecting data that meets the conditional statement. The WHERE clause can include multiple conditions.

|  |  |
| --- | --- |
| Operator | Description |
| = | Equal |
| <> | Not equal |
| > | Greater than |
| < | Less than |
| >= | Greater than or equal |
| <= | Less than or equal |
| AND | Logical And |
| OR | Logical Or |
| NOT | Logical NOT |

Example: SELECT name, hcode, class FROM student WHERE class="1A"

Result:



The LIKE Operator

The LIKE operator is used in a WHERE clause to search for a specified pattern in a column. The SQL LIKE clause is very useful when you want to specify a search condition within your SQL WHERE clause, based on a part of a column contents.

Syntax:

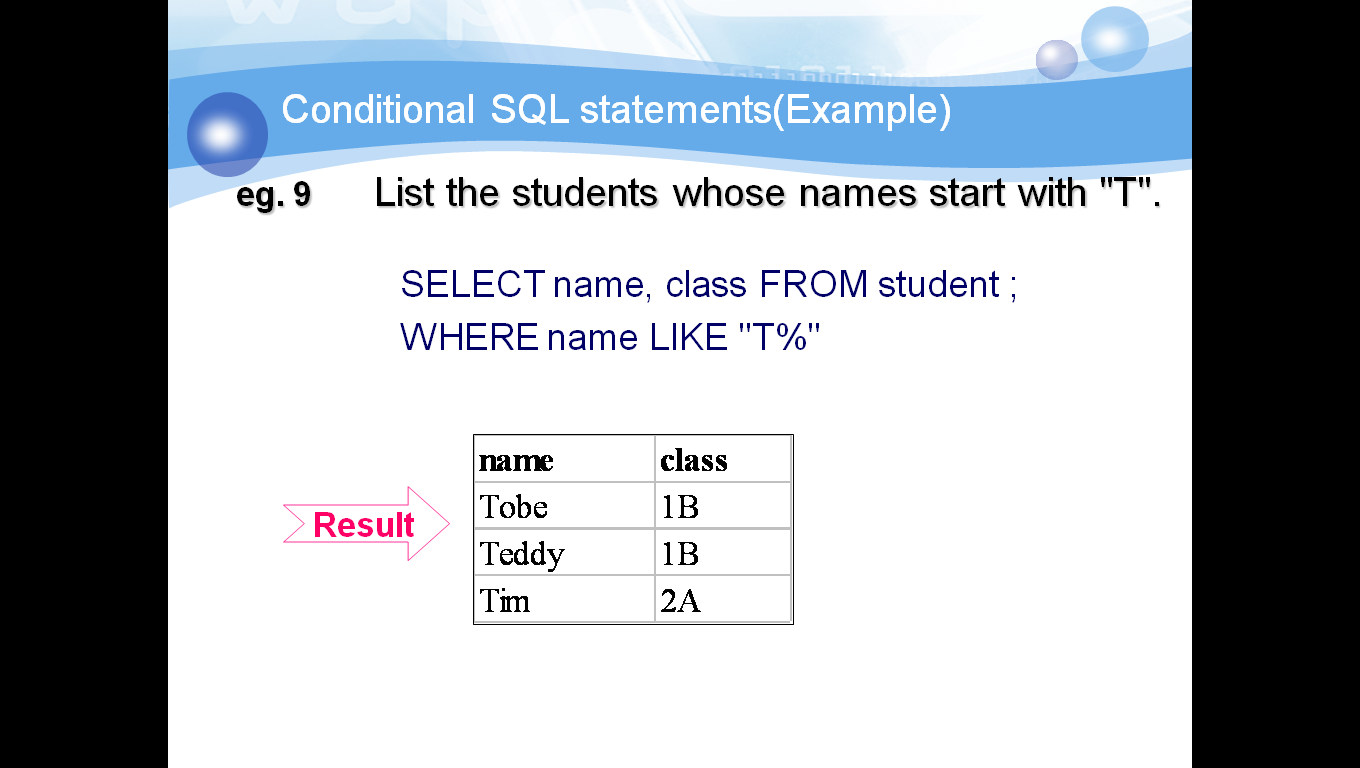
SELECT column\_name(s)  
FROM table\_name  
WHERE column\_name LIKE pattern

Example:

SELECT name, class FROM student ;

WHERE name LIKE "T%"

Result:



The '%' is a so called wildcard character and represents any string in our pattern.  
You can put the wildcard anywhere in the string following the SQL LIKE clause and you can put as many wildcards as you like too. Another wildcard character is '\_' representing any single character. The '[]' specifies a range of characters.

The BETWEEN Operator

The BETWEEN operator is used in a WHERE clause to select a range of data between two values.

Syntax:

SELECT column\_name(s)  
FROM table\_name  
WHERE column1

BETWEEN value1 AND value2

This SQL statement will return the records where column1 is within the range of value1 and value2 (inclusive). The BETWEEN function can be used in any valid SQL statement - select, insert, update, or delete.

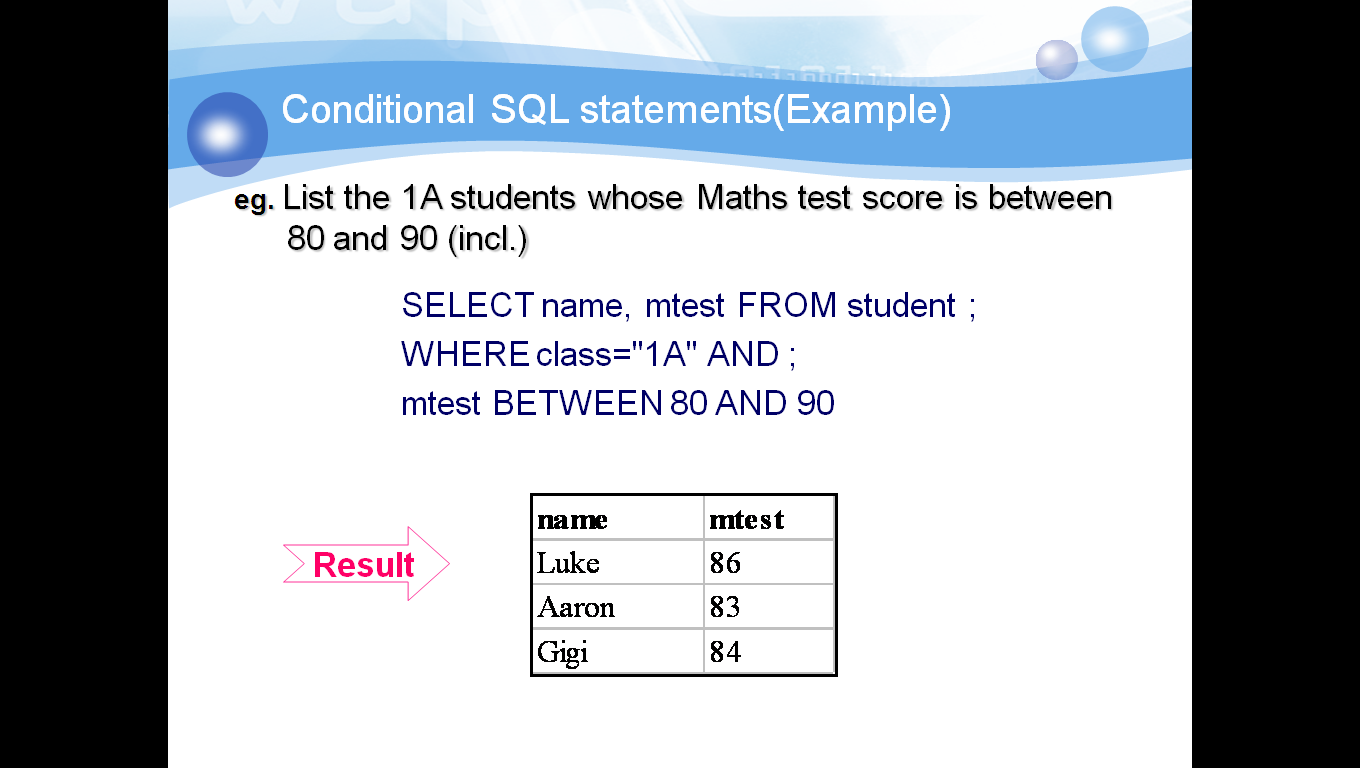
Example:

SELECT name, mtest FROM student ;

WHERE class="1A" AND ;

mtest BETWEEN 80 AND 90

Result:



**Set Membership**

The WHERE clause can be further modified to test if an attribute has a value which is member of a particular set. This is achieved using the IN condition.

Syntax:

SELECT "column\_name"  
FROM "table\_name"  
WHERE "column\_name" IN ('value1', 'value2', ...)

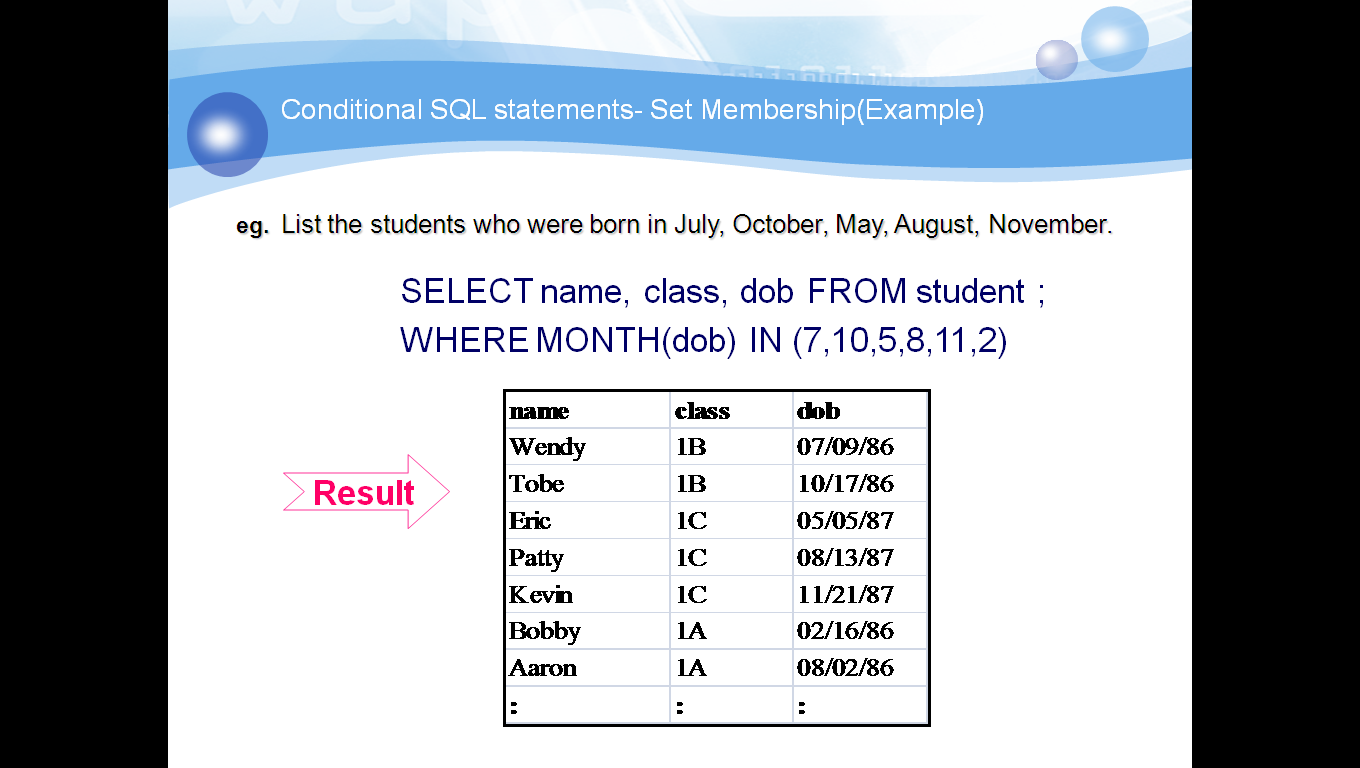
The number of values in the parenthesis can be one or more, with each values separated by comma. Values can be numerical or characters.

Example:

SELECT name, class, dob FROM student ;

WHERE MONTH(dob) IN (7,10,5,8,11,2)

Result:



# The ORDER BY Keyword

The ORDER BY keyword is used to sort the result-set by a specified column.The ORDER BY keyword sort the records in ascending order by default.If you want to sort the records in a descending order, you can use the DESC keyword. The ordering of the selected data can be done by one or more columns in a table.

Syntax:

SELECT "column\_name"  
FROM "table\_name"  
[WHERE "condition"]  
ORDER BY "column\_name" [ASC, DESC]

The [] means that the WHERE statement is optional. However, if a WHERE clause exists, it comes before the ORDER BY clause. ASC means that the results will be shown in ascending order, and DESC means that the results will be shown in descending order. If neither is specified, the default is ASC.

t is possible to order by more than one column. In this case, the ORDER BY clause above becomes

ORDER BY "column\_name1" [ASC, DESC], "column\_name2" [ASC, DESC]

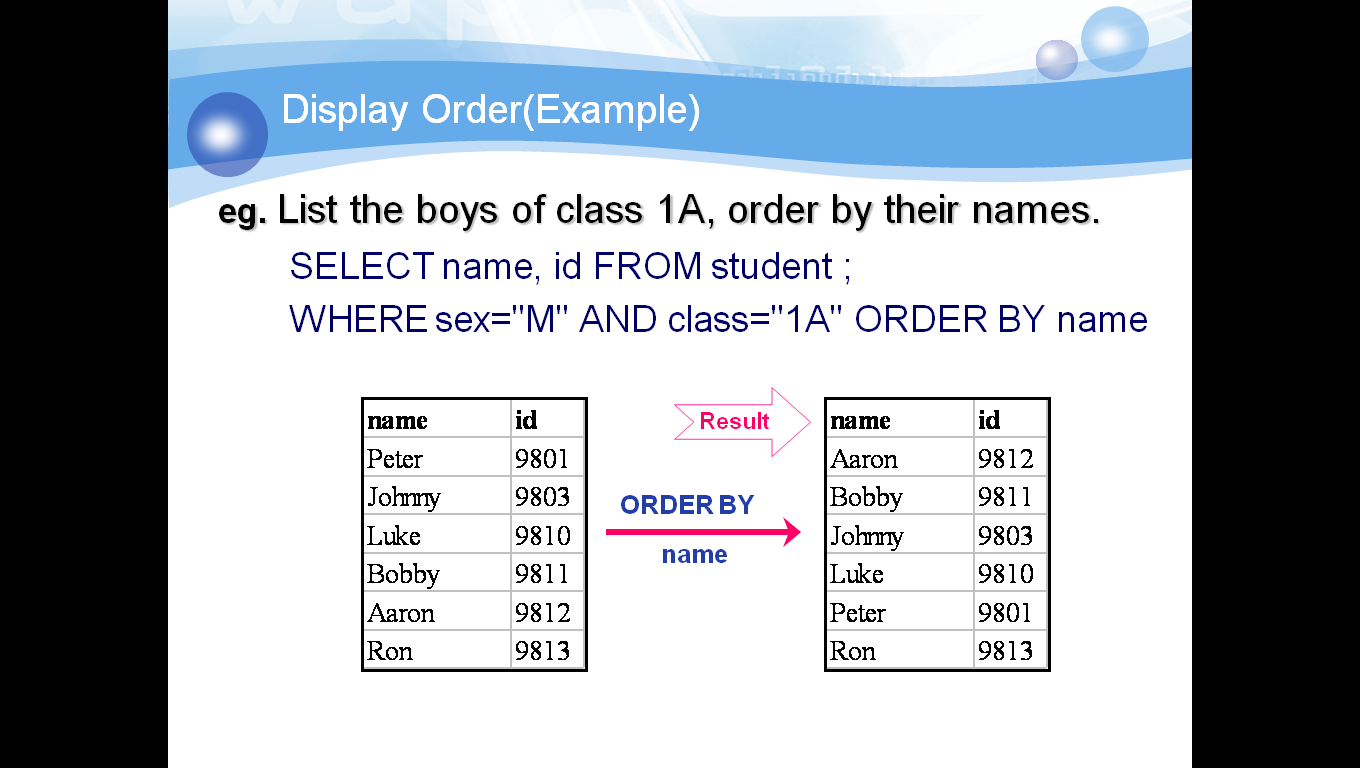
Assuming that we choose ascending order for both columns, the output will be ordered in ascending order according to column 1. If there is a tie for the value of column 1, we then sort in ascending order by column 2.

Example:

SELECT name, id FROM student ;

WHERE sex="M" AND class="1A" ORDER BY name

Result:



Aggregate Functions

SQL has several arithmetic functions, and they are:

• AVG: Average of the column.

• COUNT: Number of records.

• MAX: Maximum of the column.

• MIN: Minimum of the column.

• SUM: Sum of the column.

Syntax:

SELECT "function type" ("column\_name")  
FROM "table\_name"

In addition to using functions, it is also possible to use SQL to perform simple tasks such as addition (+) and subtraction (-).

**Grouping**

The GROUP BY clause can be used in a SELECT statement to collect data across multiple records and group the results by one or more columns. The SQL GROUP BY clause is used along with the SQL aggregate functions and specifies the groups where selected rows are placed. WHEN one or more aggregate functions are presented in the SQL SELECT column list, the SQL GROUP BY clause calculates a summary value for each group.

Syntax:

SELECT ...... FROM ...... WHERE condition ;

GROUP BY groupexpr [HAVING requirement]

groupexpr specifies the related rows to be grouped as one entry. Usually it is a column. WHERE condition specifies the condition of individual rows before the rows are group. HAVING requirement specifies the condition involving the whole group.

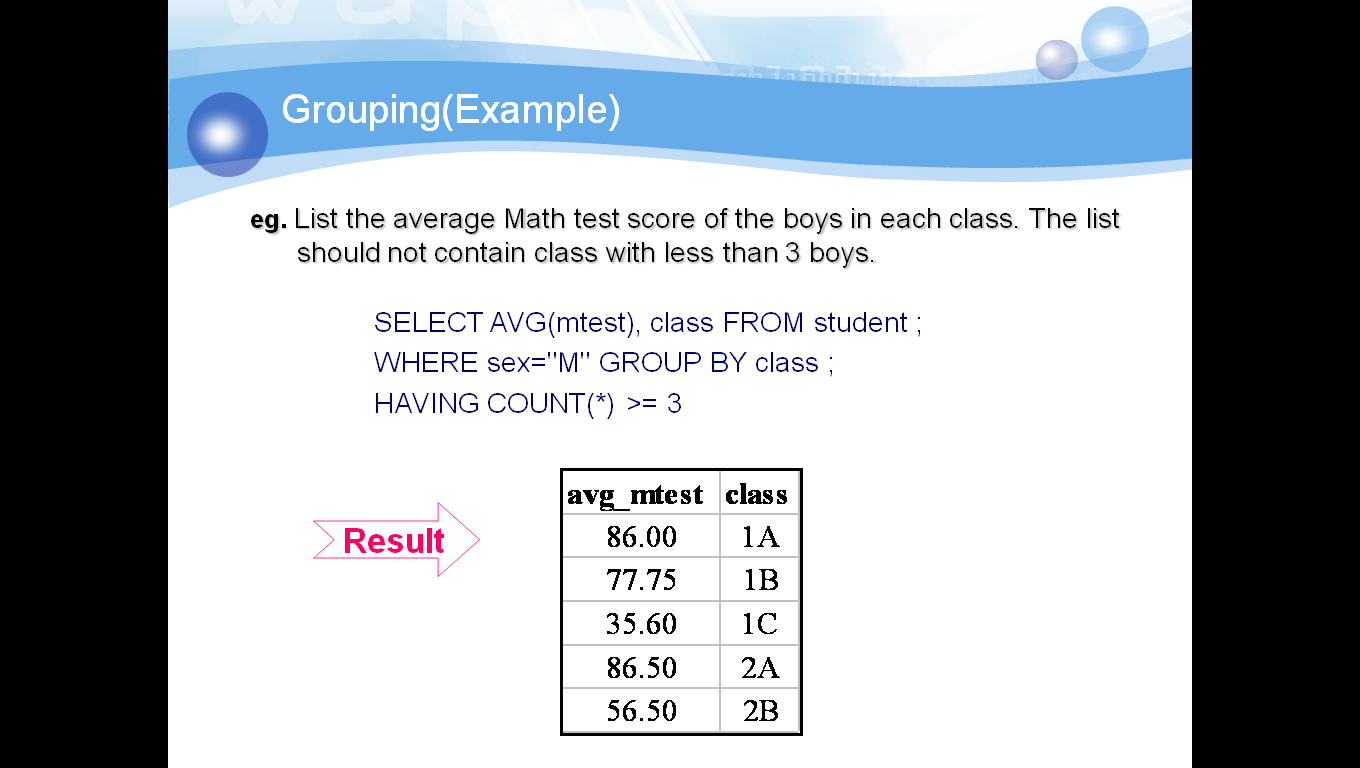
Example:

SELECT AVG(mtest), class FROM student ;

WHERE sex="M" GROUP BY class ;

HAVING COUNT(\*) >= 3

Result:



**Multiple Table Queries**

A query can return records as the result of a query across more than one relation. These multiple relation queries are described as joins or Cartesian products. SQL joins are used to query data from two or more tables, based on a relationship between certain columns in these tables. A join often realizes relationships by equating a foreign key in one relation with the primary key of another. The general form of the multiple table SQL statement can be described as follows:

* + Name two or more relations in the FROM clause.
  + Restrict the join in the WHERE clause.
  + Create a relation with all attributes of the joined relation.
  + If we remove the WHERE clause then the result is the Cartesian product

## The join operation will combine the tables into one large table with all possible combinations (Math: Cartesian Product), and then it will filter the rows of this combined table to yield useful information.

## 

## Different SQL JOINs

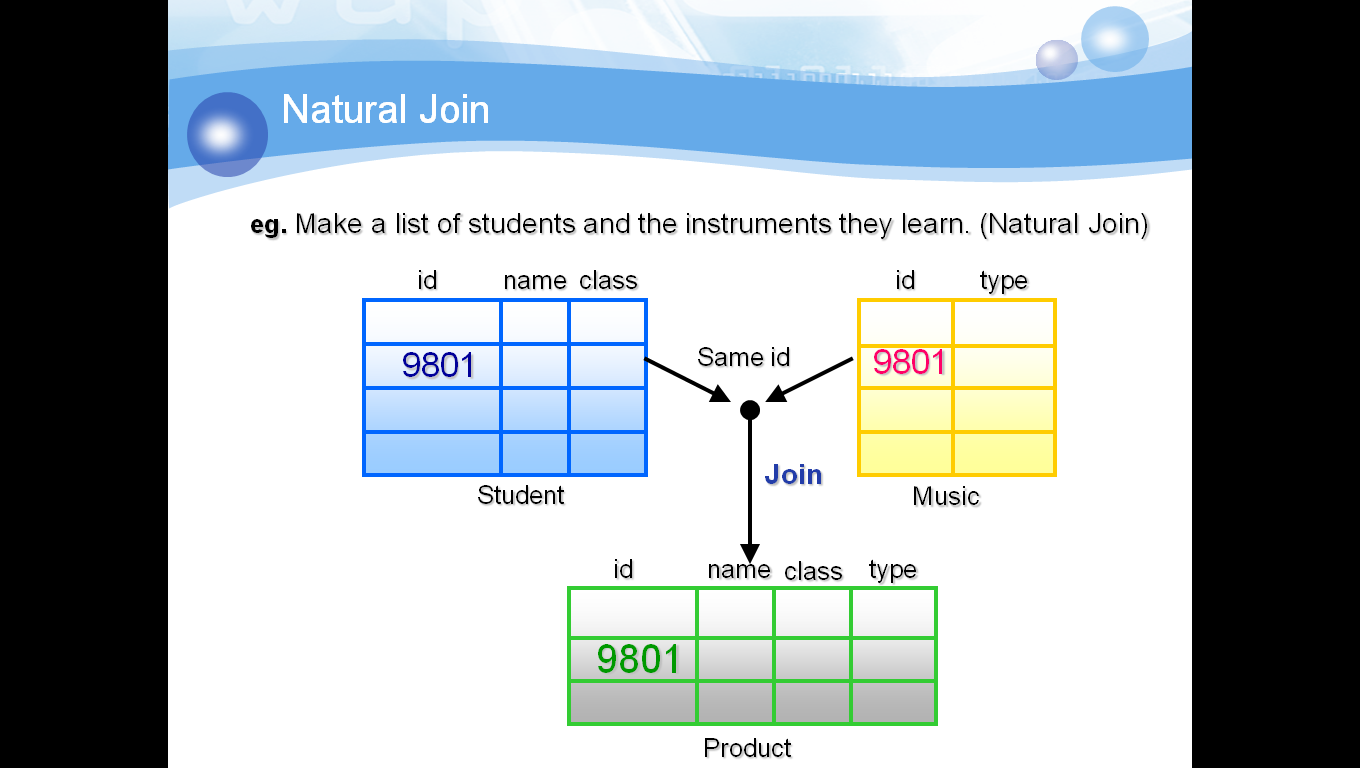
* JOIN: Return rows when there is at least one match in both tables
* LEFT JOIN: Return all rows from the left table, even if there are no matches in the right table
* RIGHT JOIN: Return all rows from the right table, even if there are no matches in the left table
* FULL JOIN: Return rows when there is a match in one of the tables

**Using Alias**

If the attribute names in the query are unique, then the relation name can be omitted from the WHERE clause. If this is not the case, then specifying the relation name is essential. This can, however, lead to long and complex SQL statements.To simplify this we can introduce an alias for the relation - this is a shorter name for the relation, which can be referred to in the WHERE clause.

Example:

Make a list of students and the instruments they learn.



SELECT s.class, s.name, s.id, m.type ;

FROM student s, music m ;

WHERE s.id=m.id ORDER BY class, name

Result:

