UNIT 03: Relational data and SQL

Lesson 3.2: SQL-99: Schema Definition, Constraints, Queries, and Views

STRUCTURE:

3.2.1. SQL-99: Schema Definition, Constraints, Queries, and Views

SQL is used to make a query to retrieve data from a Database

On receiving the request the DBMS process the request by retrieving the data from database

The process of requesting from a Database is called a Database Query and hence the name is

SQL (Structured Query Language)

SQL is a language that all commercial RDBMS implementations understand.

SQL is a non-procedural language

What Can SQL do?

SQL can execute queries against database

SQL can retrieve data from a database

SQL can insert records in a database

SQL can update records in a database

SQL can delete set of records from a database

SQL can create new databases

SQL can create tables in a database

SQL can create the stored procedures in database

SQL can create views in a database

SQL can create procedures, views and can set permissions on tables

SQL Statement

DDL (Data Definition Language)

Create, Alter, Drop, Truncate

DML (Data Definition Language)

Insert, Update, Select, Delete

DCL (Data Control Language)

Grant, Revoke, Rollback, Commit

DDL

Data Definition Language (DDL) statements are used to define the structure of the database or schema. Some examples:

CREATE - to create objects in the database

ALTER - alters the structure of the database

DROP - delete objects from the database

TRUNCATE - remove all records from the table, including all spaces allocated for the records will be removed

COMMENT - add comments to the data dictionary

RENAME - rename an object

DML

Data Manipulation Language (DML) statements are used for manipulating data within schema objects. Examples:

SELECT - retrieve data from the a database

INSERT - insert data into a table

UPDATE - updates existing data within a table

DELETE - deletes all records from the table, but the space for the records will remain

MERGE - UPSERT operation (insert or update)

CALL - call a PL/SQL or Java subprogram

EXPLAIN PLAN - explain access path to data

LOCK TABLE - control concurrency

DCL-**Data Control Language statements**.

Examples:

GRANT - gives user's access privileges to database

REVOKE - withdraw access privileges given with the GRANT command

COMMIT - save work done

ROLLBACK - restore database to original since the last COMMIT

SQL NULL Values

If a particular column in a table is optional, we can add a new record or update an existing one without adding a value to this column. This means that the field will be saved with a NULL value.

NULL values can be treated differently from other values.

NULL is used as a placeholder for unknown or inapplicable values.

SQL Working with NULL Values

Look at the following "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola		Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari		Stavanger

Suppose that the "Address" column in the "Persons" table is optional. This means that if we insert a record with no value for the "Address" column, the "Address" column will be saved with a NULL value.

How can test for NULL values?

We are not able to test for NULL values with comparison operators, such as =, <, or <>.

We have to use the IS NULL and IS NOT NULL operators.

3.2.2. Types of Constraints

SQL Constraints

They are used to specify the rules for the data in a table.

If there is any kind of violation between constraint and data action, the action is aborted by the constraint.

Constraints are specified when the table is created (inside the CREATE TABLE statement) or after the table is created (inside the ALTER TABLE statement).

SQL CREATE TABLE + CONSTRAINT Syntax

```
CREATE TABLE table_name
(
column_name1 data_type(size) constraint_name,
column_name2 data_type(size) constraint_name,
column_name3 data_type(size) constraint_name,
....
);
```

In SQL, we have the following constraints:

NOT NULL - Indicates that a column cannot store NULL value

UNIQUE - Ensures that each row for a column must have a unique value

PRIMARY KEY - A combination of a NOT NULL and UNIQUE. Ensures that a column (or combination of two or more columns) have an unique identity which helps to find a particular record in a table more easily and quickly

FOREIGN KEY - Ensure the referential integrity of the data in one table to match values in another table

CHECK - Ensures that the value in a column meets a specific condition

DEFAULT - Specifies a default value when specified none for this column

DDL - Data Definition Language

DDL is a standard subset of SQL that is used to define tables (database structure), and

Other metadata related things. The few basic commands include: CREATE DATABASE, CREATETABLE, ALTER TABLE and DROP TABLE.

CREATE DATABASE

Many database servers allow for the presence of many databases1.

In order to create a data base, a relatively standard command ‘CREATE DATABASE’ is used.

The general format of the command is:

```
CREATE DATABASE <database-name>;
```

The name shouldn’t have any spaces.

Some databases allow hyphens, and/or underscores in the name.

The name is usually limited in size.

DROP DATABASE

Drop database simply removes the data base. Note that it doesn’t ask you for confirmation, and once you remove a data base,it is gone forever.

```
DROP DATABASE <database-name>;
```

CREATE TABLE

Probably the most common DDL statement is ‘CREATE TABLE’.

It is usedto create tables.

The general format is

```
CREATE TABLE <table-name> (

...

);
```

The general format of a column definition is the column name followed by the data type.

E.g.:

```
PERSONID INT
```

This defines a column name PERSONID, of data type INT. Column names have to be comma separated,

```
CREATE TABLE PERSON (
PERSONID INT,
LNAME VARCHAR (20),
FNAME VARCHAR (20) NOT NULL,
DOB DATE,
PRIMARY KEY (PERSONID)
);
```

The above creates a table named person, with person id, first name, last name, and date of birth. There is also the ‘primary key’ definition.

A primary key is a column value that uniquely identifies a database record.

So for example, we can have two ‘person’ records with the same last name and first name, but with different ids.

Besides for primary key, there are many other flags we can specify for table columns.

For example, in the above example, FNAME is marked as NOT NULL, which means it is not allowed to have NULL values3.

DROP TABLE

It simply removes the table.

Once you remove a table, it will be removed forever.

```
DROP TABLE <table-name>;
```

ALTER TABLE

There is a command to ‘alter’ tables after you create them.

This is usually only useful if the tables have some data, and we want to modify it.

Also, most databases have some restrictions on what ‘alter table’ is allowed to do.

E.g.: Oracle allows to do add a column, but can’t remove a column.

The general syntax to add a field is:

ALTER TABLE <table-name>

ADD <field-name><data-type>

The general syntax to drop a field is:

ALTER TABLE <table-name>

DROP <field-name>

Note that very few databases let you drop a field.

The drop command is mostly present to allow for dropping of constraints.

The general syntax to modify a field (change its type, etc.) is:

ALTER TABLE <table-name>

MODIFY <field-name><new-field-declaration>

Note that you can only do this to a certain extent on most databases.

Just as with ‘drop’, this is mostly useful for working with table constraints.

DML - Data Manipulation Language

This is a standard subset of SQL that is used for data manipulation. First we need to insert data into the database. After that we can select, modify and delete it. These correspond to SELECT, INSERT, UPDATE, and DELETE statements.

INSERT Statement

To get data into a database, we use the ‘insert’ statement. The syntax is:

INSERT INTO <table-name> (<column1>,<column2>,...)

VALUES (<column-value1>,<column-value2>,..);

The column names (i.e.: column1, etc.) must correspond to column values (i.e.: column-value1,

etc.). There is a short-hand for the statement:

INSERT INTO <table-name>

VALUES (<column-value1>,<column-value2>,<column-value3>);
A typical example, of inserting the ‘student’ record in to Student table we can use following query

INSERT INTO STUDENT (STUDENTID, STUDENTNAME, DOB)

VALUES (1,’JOHN’,’1956-11-23’);

SELECT STATEMENT:

The SELECT statement is used to fetch data from a database.The result is stored in a result table, called result-set.

SQL SELECT Syntax

SELECT *column_name*,*column_name*
FROM *table_name*

WHERE <Search Condition >

The column-list indicates what columns you’re interested in (the ones which you wantto appear in the result).

Search-condition specifies what criteria we are looking for.

An example of a short-hand version to retrieve all ‘student’ records we’ve been using:

SELECT * FROM STUDENT;

SQL statement which selects the "CustomerName" , "City" columns from the "Customer" table:

SELECT CustomerName,City FROM Customers;

The WHERE Clause

The WHERE clause is used in UPDATE, DELETE, and SELECT statements.

Using WHERE clause we can retrieve the records that satisfies a specified criterion.

The syntax of where clause:

SELECT *column_name*,*column_name*
FROM *table_name*
WHERE *column_name operator value*;

UPDATE Statement

The update statement is used for changing records.

The general syntax is:

UPDATE <table-name>

SET <column1> = <value1>, <column2> = <value2>, ...

WHERE <criteria>

The criteria are what select the records for update.

The ‘set’ portion indicates which columns should be updated and to what values.

An example of the use would be:

UPDATE PERSON

SET FNAME=’Clark’, LNAME=’Kent’

WHERE FNAME=’Superman’;

Example2: Consider the customer table

```
 UPDATE Customer
SET ContactName='Alfa ', City='California'
WHERE CustomerName=' Alfreds ';
```

DELETE Statement

The ‘delete’ is used to remove elements from the database.

The syntax is very similar to update and select statements:

```
DELETE FROM <table-name>

WHERE <criteria>
```

Basically we select which records we want to delete using the where clause.

An example use would be:

```
DELETE FROM PERSON

WHERE PERSONID=12345;
```

Example2: Consider table, using the following query we can delete a row.

```
DELETE FROM Customer
WHERE CustomerName='Alfreds’ AND ContactName='Maria’;
```

- SQL Aggregate Functions

SQL aggregate functions will return a single value, calculated from values in a column.

Useful aggregate functions:

AVG() –It Returns the average value

COUNT() –It Returns the number of rows

FIRST() –It Returns the first value

LAST() –It Returns the last value

MAX() –It Returns the largest value

MIN() –It Returns the smallest value

SUM() –It Returns the sum

SQL Scalar functions

SQL scalar functions will return a single value, based on the input value.

Useful scalar functions:

UCASE() –It Converts characters to upper case

LCASE() – It Converts characters to lower case

MID() –It extracts some characters from the text field

LEN() –It Returns the length of a text field

ROUND() –It Rounds a numeric field to the number of decimals specified

NOW() –It Returns the current system date and time

FORMAT() –It Formats how a field is to be displayed

DCL - Data Control Language

This is used for security management.

There exist two commands: GRANT, and REVOKE,

GRANT Statement

The general syntax is like this:

GRANT <privilege> ON <object> TO <who>;

Basically, a privilege can be something like ‘update’ or ‘select’, etc., or it can be ‘all’ when granting ‘all’ privileges.

An ‘object’ can be anything, but is often a database, or database table.

The 'who' is generally a database login/ user.

Some databases (like MySQL) will actually create the user if they don't already exist.

In some cases, we also have the option for specifying the user password, via: 'identified by' parameter.

REVOKE Statement

The revoke is the opposite of GRANT. The general syntax is:

REVOKE <privilege> ON <object> FROM <who> ;

- Different SQL JOINs

INNER JOIN: It returns all the rows when there is at least one matched column in BOTH tables. Refer the previous examples for inner join.

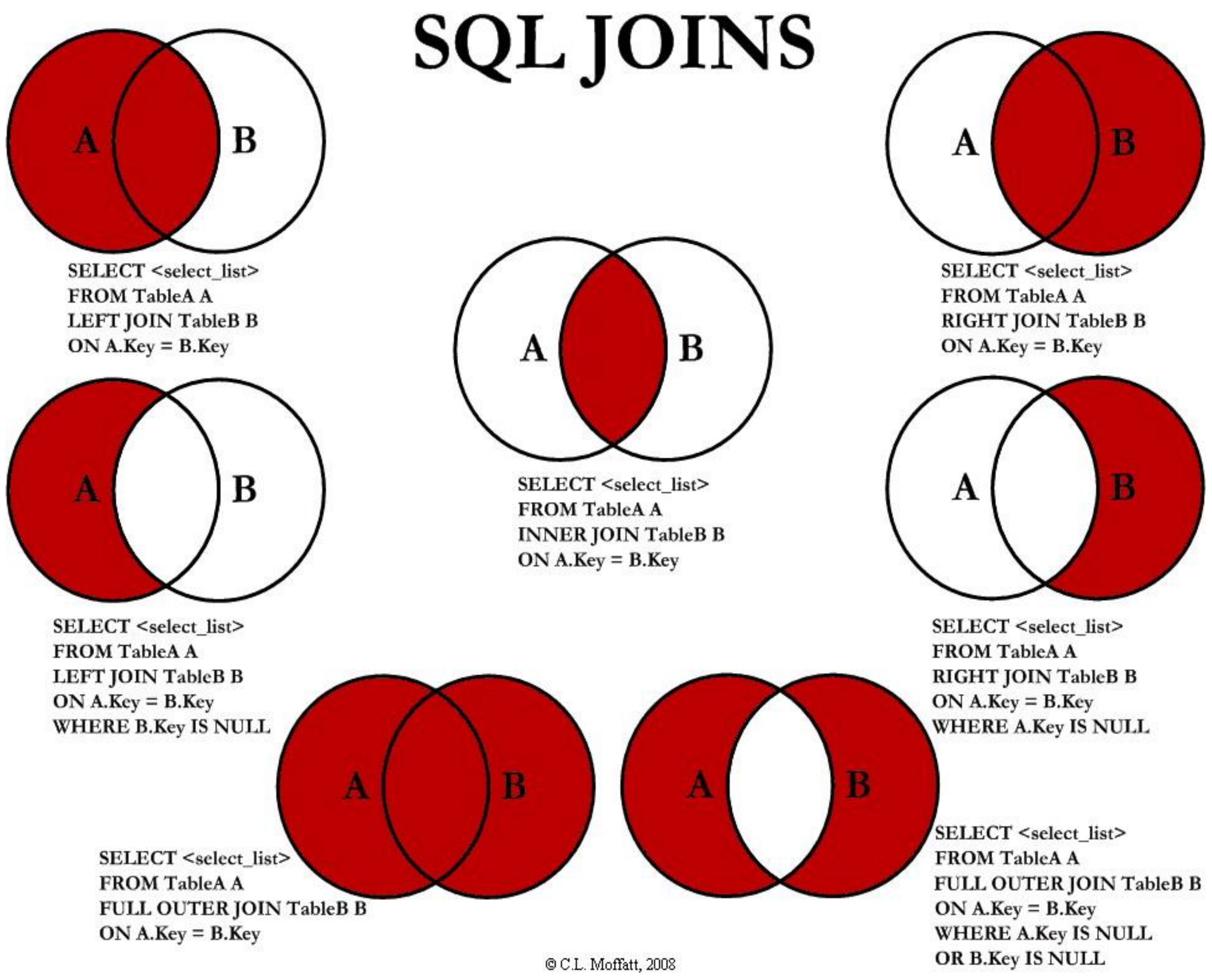
LEFT JOIN: It returns all rows from the left table, and only the matched rows from the right table. Refer the previous examples for left join.

RIGHT JOIN: It returns all rows from the right table, and only the matched rows from the left table. Refer the previous examples for Right join.

FULL JOIN: Return all rows when there is a match in ONE of the tables. Refer the previous examples for full join.

Different types of SQL join operations are shown in the following figure 2.7

The following figure



```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```

```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```

```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```

```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
```

```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```

```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
```

Figure 2.7

- Bitmap Index

A bitmap index is some special type of index that is mainly used in the Oracle database.

Bitmap indexes are considered to be used on *low-cardinality columns*. A low cardinality column means that the column has relatively few unique values. For example, a column called Sex which has only “Male” and “Female” as the two possible values is considered low cardinality because there are only two unique values in the column. The extreme case of low cardinality is Boolean data which has two values, True and False.

Bitmap indexes use bit arrays (called bitmaps) and answer queries by performing bitwise logical operations on these bitmaps.

An example of a bitmap index

Consider a table called People in which each row is a different person, it contains columns like Name, Address, etc. Let's set a column called SexualOrientation in the people table, and the three possible values for this column are heterosexual, homosexual, and bisexual. In this case,when we create a bitmap index on this column, then for each unique value in the SexualOrientation column, a separate index record will be created. So, there will be one index record for the "heterosexual" value, an index record for the "homosexual" value, and the third index record for the "bisexual" index record. All three of these index records will be the part of the bitmap index

Now we have 3 index records – one for each unique value in the Sexual Orientation column. Each record will have the same number of bits as the number of rows in the table.

Let's say that there are 1,000 rows in the People table. Then, each of these records will have 1,000 bits – one for each row in the table ***.Every bit corresponds to a row in the table***. A binary value of '1' means that the particular index record has it's corresponding row is set to the value that index record represents. A '0' means that the value represented by the index record is not set in that row's column. So, let's say that in the People table the 10th row has the Sexual Orientation column is set to "heterosexual" and the 11th row has the SexualOrientation column set to "homosexual". This means that the index record for "heterosexual" will have its 10th bit set to 1 and it's 11th bit set to "0", because the 10th row contains the value "heterosexual" – so the "1" indicates that there is a match. The 0 indicates that there is no match, but the index record for homosexual will have a "1" in the 11th row, and a 0 in the 10th row.

Example of how to create a bitmap index

Here is the SQL that can be used to create a bitmap index:

```
CREATE BITMAP INDEX IX_PEOPLE_SEXUAL_ORIENTATION

ON PEOPLE (SEXUALORIENTATION);
```

- Assertions and Triggers

A trigger is a piece of SQL code which executes either before or after an update, insert, or delete in the database. Each trigger is attached to a single, specified table in the database.

The difference between assertions and checks is a little murkier; many databases don't even support assertions.

Check Constraint - A check is a piece of SQL code which ensures a condition is satisfied before an action can be applied on a record. Ex: All customers must have an account balance of at least $100 in their account.

```
ALTER TABLE accounts

ADD CONSTRAINT CK_minimumBalance

CHECK (balance >= 100)
```

Assertions - An assertion is a piece of SQL code which ensures a condition is satisfied or it stops action being taken on a **database object**.

An assertion is a Boolean-valued SQL expression that will be always true.

An **assertion** is a predicate expressing a condition we expect the database to always satisfy.

Domain constraints, functional dependency and referential integrity are special types of assertion.

Where a constraint cannot be expressed in the above forms, we can use an assertion. When an assertion is created, the system tests it for validity. If the assertion is valid, any further modification to the database is allowed only if it does not cause that assertion to be violated. This testing may result in significant overhead if the assertions are complex.

So the **assert** should be used with great care.

A trigger is a series of actions associated with some set of events. Assertions are easy to define, and difficult to implement. The DBMS must determine when modifications affect an assertion’s truth - declarative Triggers specifically identify what the DBMS needs to do - procedural

CREATE ASSERTION CHECK () -The condition must be true when the assertion is created and must remain true

Example:

CREATE ASSERTION TooRich CHECK ((NOT EXISTS (SELECT company-name FROM Company, Executives WHERE ceo = eid AND networth < 10000000)); Ensure that no company has as its CEO someone whose net worth is less than $10,000,000. Triggers are also known as event-condition-action or ECA rules.

1 Triggers are awakened by the programmer specified events, eg insert, delete or update on a relation or on a transaction end.

2 Triggers test a condition. If the condition does not hold, nothing else is done.

3 If the condition is satisfied, the corresponding action will be performed. It can modify the effects of the event, or even abort the transaction. It can perform any sequence of database operations that may have nothing to do with the particular event.

Features of an SQL trigger are

1 The check of the condition and the action may be executed on the database state before the triggering event is executed, or the state that exists after the event is executed.

2 The condition an action can refer to the old and the new values of tuples after an update.

3 It is possible to limit update events to a certain attribute or set of attributes.

4 A trigger may execute once for each modified tuple (row-level trigger) or once for all tuples that are changed by an SQL statement (statement-level trigger).

Example:

CREATE TRIGGER NetWorthTrigger AFTER UPDATE of netWorth on MovieExec REFERENCING OLD ROW as OldTuple, NEW ROW as NewTuple FOR EACH ROW WHEN (OldTuple.netWorth > NewTuple.netWorth) UPDATE MovieExec SET netWorth = OldTuple.netWorth WHERE cert = NewTuple.cert;

This trigger is designed to stop (by undoing) any update on an attribute (netWorth) with a lower value. There is limited support for triggers in MySQL, but restricted to users with SUPER privileges. As of Version 5.1.6 triggers can be created and dropped when appropriate privileges are set.

- SQL Views

View is a table that is derived from the actual table.

It is the result set of a *stored* query on the data.

The database users can query just as they would in a persistent database collection object.

The ordinary *base tables* in a relational database.

It is a virtual table computed or collated dynamically from data in the database when access to that view is requested.

Views provide some advantages over tables:

Views represent a subset of the data of the actual table.

Views can join and simplify multiple tables into a single virtual table.

Views can act as aggregated tables, where the database engine aggregates data (sum, average, etc.) and presents the calculated results as part of the data.

Views can hide the complexity of data.

Views require less space for storage; the database contains only the definition of a view, not a copy of all the data.

Depending on the SQL engine used, views can provide extra security.

SQL CREATE VIEW Statement

In SQL, a view is a virtual table based on the result-set of an SQL statement.

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.

SQL CREATE VIEW Syntax:-

```
CREATE VIEW view_name AS
SELECT column_name(s)
FROM table_name
WHERE condition
```

SQL CREATE VIEW Examples

If you have the corporate database you can see that it has several views installed by default.

The view "Current Product List" lists all active products (products that are not discontinued) from the "Products" table.

```
CREATE VIEW [Current Product List] AS
SELECT ProductID,ProductName
FROM Products
WHERE Discontinued=No
```

We can query the above view as follows:

```
SELECT * FROM [Current Product List]
```

SQL Updating a View

We can update a view by using the following syntax:

SQL CREATE OR REPLACE VIEW Syntax:-

```
CREATE OR REPLACE VIEW view_name AS
SELECT column_name(s)
FROM table_name
WHERE condition
```

Now we want to add the "Category" column to the "Current Product List" view. We will update the view with the following SQL:

```
CREATE VIEW [Current Product List] AS
SELECT ProductID,ProductName,Category
```

```
FROM Products
WHERE Discontinued=No
```

SQL Dropping a View

We can delete a view with a DROP VIEW command.

SQL DROP VIEW Syntax

```
DROP VIEW view_name
```

Comparison between Tables and views

Tables	**Views**
A table is a database object which is used to hold data that are used in reports and applications	A view is a single table which is derived from another table.
A table contains limited number of columns and an unlimited number of rows. The tuples are stored in the actual database.	View is a virtual table that does not necessarily exist in the physical form.
Several tables are needed to store linked data and records.	A view can incorporate several tables into one virtual table
A view is used to query certain data contained in different tables	Table holds basic user data and holds instances of a defined object.
The tables hold the data in the actual database	Frequently queried data can be accessed through view, and updating the data in the database requires updation on the data shown in the view also.

3.2.3. Embedded SQL

Embedded SQL is a method of inserting SQL commands or queries into the code of a programming language, which is known as a host language. The host language is not able to parse SQL.So the inserted SQL will be parsed by the embedded SQL pre-processor.

Embedded SQL is robust and convenient method of combining the computing power of a programming language with SQL's specialized data manipulation capabilities.

Any host language variables that are passed in to the SQL command must be declared in SQL. Here it may come two serious situations. The data types recognised by SQL may not be recognised by the host language. This can be addressed by type casting of data values before passing in to the SQL commands. Another has to do with SQL being set oriented, and is addressed using cursors. The C programming language is commonly used for embedded SQL implementation. Embedded SQL was once the predominant standard way to mix SQL statements with host languages. It lets you mix SQL statements directly into an application program written in some common computer programming language.

The host language variables must be prefixed by a colon. And the declaration would be like the following:

EXEC SQL BEGIN DECLARE SECTION

Long c_id;
char c_name;

Float c_type;

EXEC SQL END DECLARE SECTION

The syntax and use of embedded SQL are described below.

‘&sql’

Embedded SQL statements are started from the rest of the code by the &sql() directive, as shown in the following example:

```
  WRITE "Invoke  Embedded SQL",!
  &sql(SELECT Name INTO :a FROM Sample.Person)

  WRITE "The SQL error code ",SQLCODE,!

  WRITE "The name ",a
```

Results are returned using the INTO clause and host variables. In this case, the host variable is named as :a.

Embedded SQL is not supported by all RDBMS. Oracle Database and Postgre SQL provide embedded SQL support. MySQL, Server 2008, Sybase and SQL do not. For example, a commercial bank's information system has a front-end user interface created in the C language, and the back-end is Oracle DB database. One of the front-end interface modules allows quick viewing and commission calculation for sales agents during specified periods. An inefficient approach to handling this process would be to store each commission value in a database table. However, a more effective solution is to calculate and return commission values based on unique user requests on specified dates.

The application accomplishes this by embedding a SQL query within the C code like

```
SELECT 0.2*SALE_AMOUNT FROM TOTAL_SALES WHERE SALE_DATE='MM/DD'YYYY' AND AGENT_NO=xx
```

Here the SQL statement calculates and returns 20 percentage of the sale amount from a TOTAL_SALES table, while the user is expected to input the SALE_DATE and AGENT_NO values. This SQL query is then inserted inline into the C code of the front-end module. The C code and SQL query will work together to deliver good user results. Each embedded SQL statement introduced with EXEC SQL.

3.2.4. 4GLs

4GL programming language often abbreviated *4GL,* was born in Informix corp 15 years ago. Fourth-generation languages are programming languages closer to human languages than typical high-level programming languages. Most 4GLs are used to access databases.

4GLs are associated with databases and data processing, allowing the efficient development of business-oriented systems with languages that are closely match the way domain experts formulate business rules and processing sequences. Many of such data-oriented 4GLs are based on the SQL, invented by IBM and then adopted by ANSI and ISO as the standard language for managing structured data.

Most 4GLs contain the ability to add 3GL-level code to introduce specific system logic into the 4GL program.

The most determined 4GLs, is denoted as Fourth Generation Environments, attempt to produce entire systems from a design made in CASE tools and the additional specification of data structures, reports ,screens and some specific logic.

An example of a query would be as follows:

FIND ALL RECORDS WHERE NAME IS "RAM"

3.2.5. Procedural extension to SQL

Basic Structure of PL/SQL

PL/SQL stands for Procedural Language/SQL. PL/SQL extends SQL by introducing constructs found in procedural languages, produce in a structural language that is more powerful than SQL. The basic unit in PL/SQL is a block. All PL/SQL programs are made up of blocks, and can be nested within each other. Each block performs a logical action in the program.

A PL/SQL block is defined by the keywords DECLARE, BEGIN, EXCEPTION, and END. These keywords divide the block into a declarative part, an executable part, and an exception-handling part. The declaration section is optional and may be used to define and initialize constants and variables. If a variable is not initialized then it defaults to NULL value. The optional exception-handling part is used to handle run time errors. Only the executable part is required. A block can have a label.

A block has the following structure:

```
DECLARE

/* Declarative section: variables, types, and local subprograms. */

BEGIN
```

```
/* Executable section: procedural and SQL statements go here. */

/* This is the only section of the block that is required. */

EXCEPTION

/* Exception handling section: error handling statements go here. */

END;
```

Consider an Example:

```
<<label>>   -- optional
declare
-- optional
  number1 number(2);
  number2 number1%type    := 17;            -- value default
  text1   varchar2(12) := 'Hello world';
  text2   date        := SYSDATE;       -- current date and time
begin
-- this section is mandatory, must contain at least one executable statement
  SELECT street_number
    INTO number1
    FROM address
    WHERE name = 'INU';
exception
```

```
-- this section is optional

  WHEN OTHERS THEN

    DBMS_OUTPUT.PUT_LINE('Error Code is ' || to_char(sqlcode )  );

    DBMS_OUTPUT.PUT_LINE('Error Message is ' || sqlerrm   );

end;
```

The symbol := functions as an assignment operator to store a value in a variable.

Blocks can be nested i.e. because a block is an executable statement; it can appear in another block wherever an executable statement is allowed.

The only SQL statements allowed in a PL/SQL program are SELECT, INSERT, UPDATE, DELETE ,several other data manipulation statements and some transaction control. The SELECT statement has a special form in which a single tuple is placed in variables. Data definition statements like CREATE, DROP, or ALTER are not allowed here. The executable section also contains constructs such as assignments, branches, loops, procedure calls and triggers.PL/SQL is not case sensitive. C style comments (/* ... */) can be used.

To execute a PL/SQL program, we must follow the program text itself by

* A line with a single dot ("."), and then
* A line with run;

Variables and Types

Communication between a PL/SQL program and the database is through the variables. Every variable has a specific type associated with it. That type can be
* A generic type used in PL/SQL such as NUMBER
* One of the types used by SQL for database columns
* Declared to be the same as the type of some database column

Variables of type NUMBER can hold either an integer value or a real number. The most commonly used character string type is VARCHAR(n) and commonly used generic type is NUMBER, where n is the maximum length of the string in bytes. The length is required, and there is no default.

For example, we might declare:

```
DECLARE

price NUMBER;

myBeer VARCHAR(20);
```

Note that PL/SQL allows BOOLEAN variables, although Oracle does not support BOOLEAN as a type for database columns.
A PL/SQL variable is used to manipulate data stored in a existing relation. Here it is essential that the variable have the same type as the relation column. If there is any type mismatch, variable assignments and comparisons may not work the way we expect. To be safe, instead of hard coding the type of a variable, we should use the %TYPE operator.

For example:

```
DECLARE

myBeer Beers.name%TYPE;
```

gives PL/SQL variable myBeer whatever type was declared for the name column in relation Beers.

A variable may also have a type that is a record with several fields. The simplest way to declare such a variable is to use %ROWTYPE on a relation name. The result is a record type in which the fields have the same names and types as the attributes of the relation.

For Example

```
DECLARE
```

beerTuple Beers%ROWTYPE;

makes variable beerTuple be a record with fields name and manufacture, assuming that the relation has the schema Beers(name, manufacture).

The initial value of any variable is NULL. We can assign values to variables, using the ":=" operator. The assignment occurs either immediately after the type of the variable is declared, or anywhere in the executable portion of the program.

Example:

```
DECLARE

a NUMBER := 3;

BEGIN

a := a + 1;

END;

run;
```

This program has no effect when executing, because there are no changes to the database.

Function

The purpose of a PL/SQL function is to compute and return a single value. The returned value may be a single scalar value (such as a number, date or character string) or a single collection (such as a nested table).

The PL/SQL function has the format:

CREATE OR REPLACE FUNCTION <function_name> [(**input**/**output variable** declarations)] **RETURN** return_type

```
[AUTHID <CURRENT_USER | DEFINER>] <IS|AS>   -- heading part

amount number;   -- declaration block

BEGIN   -- executable part

        <PL/SQL block with return statement>

    RETURN <return_value>;

[Exception

        none]

    RETURN <return_value>;

END;
```

3.2.6. Introduction to Query-by-example (QBE)

QBE is developed by M.M. Zloof at the IBM Yorktown Heights Laboratory. Query-by-Example is another language for querying relational data. It is having a graphical user interface that allows users to write queries by creating example tables on the screen. A user needs minimal information to get started and the whole language contains relatively few concepts. QBE is especially suited for queries that are not too complex and can be expressed in terms of a few tables.

Query-By-Example (QBE) is the first interactive database query language to exploit such modes of HCI. In QBE, a query is a construction on an interactive terminal involving two-dimensional 'drawings' of one or more relations, visualised in tabular form, which are filled in selected columns with 'examples' of data items to be retrieved. The system answers the query by fetching data items based on the given example and drawing the result on the same screen

Variables and Constants filling out a table with an example, the simplest item that can be entered under a column is a free variable or a constant. The free variable in QBE must be an identifier but a constant can be a number, string or other constructions denoting a single data value. A query containing combinations of free variables and constants is a request for a set of values instantiating the specified variables while matching the constants under the specified columns.

QBE Query and

Its results

Customer Example		
cid	cname	ccountry
p.x	p.y	india

Customer Result		
cid	cname	ccountry
1	Ram	India
2	Smith	india

pid	pname	Price
p.a	p.b	

Print/display

Here Two variables are introduced in the query: a and b. By placing a variable under a column, we are in effect assigning that variable to range over the domain of that column. Thus, the variable a ranges over the domain Pid while b ranges over Pname. Also note that the variables are prefixed by **“P.”.** In QBE, it is required if the instantiation found for the specified variable is to be displayed, ie. the prefix “P.” may be considered as a command to print. We can say that if neither variable in above figure was preceded by “P.” then the output table would display nothing.