Introduction to QUEL

QUEL is a relational database query language, based on tuple relational calculus, with some similarities to SQL.

- QUEL stands for Query Language.
- It is a data definition and data manipulation for INGRES.
- INGRES stands for Interactive Graphics and Retrieval System.
- INGRES is a relational database management system developed by Michael Stonebraker.
- QUEL does not support relational algebraic operations such as intersection, minus or union.
- It is based on tuple calculus and does not support nested sub queries.

Data Definition in QUEL

Following are the data definition statements used in QUEL,

1. CREATE

It is used to create tables or relations.

Syntax:

CREATE <table-name> <list-of-column-name>

2. RANGE

It allows to declare a range variable and restricts to assume the values that are rows from the specified table. Row variables are called Range variables in QUEL.

Syntax:

RANGE OF <variable-name> IS <relation-name>

3. INDEX

It is used to specify the name of the secondary index to be built and the columns in the table on which the index is to be created.

Syntax:

```
INDEX ON <table-name> IS <index-name> (column-name [, column-name, ...])
```

4. DESTROY

It is used to eliminate a table, index or view.

Syntax:

```
DESTROY name [, name, ...]
```

5. MODIFY

It is used to modify the structure of a table. The storage structure supported by INGRES are B-tree, hash, ISAM and heap. The storage structure will be modified from the current one to the one specified in the statement.

Syntax:

```
MODIFY <table-name>
TO <storage-structure>
ON <column-name>
```

Types of QUEL

There are *interactive* and *embedded* releases of QUEL.

- Interactive QUEL enables you to enter QUEL statements from a terminal and display query results on the terminal screen.
- Embedded QUEL enables you to include QUEL statements in programs written in programming languages such as C or Fortran.

INTERACTIVE QUEL

There are two ways to use interactive QUEL:

- The forms-based interactive terminal monitor is invoked by the iquel command. Enter QUEL statements into a form and select commands from a menu line.
- The command-based Terminal Monitor is invoked by the quel command.

EMBEDDED QUEL

Embedded QUEL (EQUEL) enables you to include QUEL statements in application programs.

For each host language, there is an EQUEL preprocessor. The preprocessor scans your source code for QUEL statements and translates the QUEL statements into the appropriate host language statements

Embedded QUEL offers following features:
Database cursors and transaction processing
Database cursors enable the application to process database rows that fulfill specified search criteria. Transactions help to preserve database integrity by grouping QUEL statements; if a transaction fails for any reason, the effects of all the statements in the transaction are undone.

Dynamic programming
application program can specify portions of many QUEL statements using host variables. The param statement enables database manipulation statements to be built dynamically, in cases where the number and data type of objects to be operated on is not determined until runtime.

Status information
QUEL provides inquiry statements that return detailed information about the database and forms being used by application program.

Runtime error handling
In EQUEL applications, we can silence error messages and trap errors using an error handler routine.

Repeat queries
we can reduce the overhead required to run an embedded query that is executed many times by using repeat queries. The first time a repeat query is executed, the DBMS Server encodes the query. On subsequent executions of the query, this encoding can account for significant performance improvements.

Elements of QUEL Statements
•Functions, operators, and predicates
•Arithmetic operations, assignments, and other basic operations
•Expressions and search conditions in queries

Difference between QUEL and SQL

QUEL	SQL
`create student(name = c10, age = i4, sex = c1, state = c2)` `range of s is student` `append to s (name = "philip", age = 17, sex = "m", state = "FL")` `retrieve (s.all) where s.state = "FL"` `replace s (age=s.age+1)` `retrieve (s.all)` `delete s where s.name="philip"`	`create table student(name char(10), age int, sex char(1), state char(2));` `insert into student (name, age, sex, state) values ('philip', 17, 'm', 'FL');` `select * from student where state = 'FL';` `update student set age=age+1;` `select * from student;` `delete from student where name='philip';`