ST. XAVIER’S COLLEGE

**Maitighar, Kathmandu**

**(Affiliated to Tribhuvan University)**



**Database Management System**

**Theory Assignment #8**

**Submitted By**

Alok Shrestha

013BSCIT005

B.Sc. CSIT Year II/IV Semester

**Submitted To**

Er. Sanjay Kumar Yadav

Lecturer

Department of Computer Science

St. Xavier’s College

Maitighar, Kathmandu

**Submitted On**

September 2015

# Background

SQL has become the standard relational database language. It has several parts:

* Data definition language (DDL) - provides commands to
  + Define relation schemes.
  + Delete relations.
  + Create indices.
  + Modify schemes.
* Interactive data manipulation language (DML) - a query language based on both relational algebra and tuple relational calculus, plus commands to insert, delete and modify tuples.
* Embedded data manipulation language - for use within programming languages like C, PL/1, Cobol, Pascal, etc.
* View Denition - commands for dening views
* Authorization - specifying access rights to relations and views.
* Integrity - a limited form of integrity checking.
* Transaction control - specifying beginning and end of transactions.

# Data-Definition Language

The SQL DDL (Data Definition Language) allows specification of not only a set of relations, but also the following information for each relation:

* The schema for each relation.
* The domain of values associated with each attribute.
* Integrity constraints.
* The set of indices for each relation.
* Security and authorization information.
* Physical storage structure on disk.

**Domain Types in SQL**

1. The SQL-92 standard supports a variety of built-in domain types:
   * **char**(n) (or **character**(n)): fixed-length character string, with user-specified length.
   * **varchar**(n) (or **character varying**): variable-length character string, with user-specified maximum length.
   * **int** or **integer**: an integer (length is machine-dependent).
   * **smallint**: a small integer (length is machine-dependent).
   * **numeric**(*p, d*): a fixed-point number with user-specified precision, consists of *p* digits (plus a sign) and *d* of *p* digits are to the right of the decimal point. E.g., **numeric**(*3, 1*) allows 44.5 to be stored exactly but not 444.5.
   * **real** or **double precision**: floating-point or double-precision floating-point numbers, with machine-dependent precision.
   * **float**(n): floating-point, with user-specified precision of at least *n* digits.
   * **date**: a calendar date, containing four digit year, month, and day of the month.
   * **time**: the time of the day in hours, minutes, and seconds.
2. SQL-92 allows arithmetic and comparison operations on various numeric domains, including, **interval** and *cast* (*type coercion*) such as transforming between *smallint* and *int*. It considers strings with different length are compatible types as well.
3. SQL-92 allows **create domain** statement, e.g.,

**create domain** *person-name* **char**(20)

**Schema definition in SQL**

1. An SQL relation is defined by:

**create table** *r* ( tex2html_wrap_inline1854

*integrity- tex2html_wrap_inline1856*  ,

..., *integrity- tex2html_wrap_inline1856*  )

where *r* is the relation name, tex2html_wrap_inline1730 is the name of an attribute, and tex2html_wrap_inline1864 is the domain of that attribute. The allowed integrity-constraints include

**primary key** tex2html_wrap_inline1866

and

**check(*P*)**

1. Example.

**create table** *branch* (

bname **char**(15) **not null**

bcity **char**(30)

assets **integer**

**primary key** (*bname*)

**check** (*assets >= 0*))

1. The values of primary key must be *not null* and *unique*. SQL-92 considers **not null** in primary key specification is redundant but SQL-89 requires to define it explicitly.
2. Check creates type checking functionality which could be quite useful. E.g.,

**create table** *student* (

*name* **char**(15) **not null**

*student-id* **char**(10) **not null**

*degree-level* **char**(15) **not null**

**check** (*degree-level* **in**

(``Bachelors'', ``Masters'', ``Doctorate'')))

1. Some checking (such as *foreign-key* constraints) could be costly, e.g.,

**check** (*bname* **in** (**select** *bname* **from** *branch*))

1. A newly loaded table is empty. The **insert** command can be used to load it, or use special bulk loader utilities.
2. To remove a relation from the database, we can use the **drop table** command:

**drop table** *r*

This is not the same as

**delete** *r*

which retains the relation, but deletes all tuples in it.

1. The **alter table** command can be used to add or drop attributes to an existing relation *r*:

**alter table** *r* **add** *A* *D*

where *A* is the attribute and *D* is the domain to be added.

**alter table** *r* **drop** *A*

where *A* is the attribute to be dropped.

**Data Manipulation Language**

# Basic Structure

1. Basic structure of an SQL expression consists of **select, from** and **where** clauses.
   * **select** clause lists attributes to be copied - corresponds to relational algebra **project**.
   * **from** clause corresponds to Cartesian product - lists relations to be used.
   * **where** clause corresponds to selection predicate in relational algebra.
2. Typical query has the form

**select** tex2html_wrap_inline1726

**from** tex2html_wrap_inline1728

**where** P

where each tex2html_wrap_inline1730 represents an attribute, each tex2html_wrap_inline1732 a relation, and *P* is a predicate.

1. This is equivalent to the relational algebra expression

displaymath1724

* + If the where clause is omitted, the predicate *P* is true.
  + The list of attributes can be replaced with a \* to select all.
  + SQL forms the Cartesian product of the relations named, performs a selection using the predicate, then projects the result onto the attributes named.
  + The result of an SQL query is a relation.
  + SQL may internally convert into more efficient expressions.

**The select Clause**

1. An example: Find the names of all branches in the *account* relation.

**select** *bname*

**from** *account*

1. **distinct** vs. **all**: elimination or not elimination of duplicates.

Find the names of all branches in the *account* relation.

**select distinct** *bname*

**from** *account*

By default, duplicates are not removed. We can state it explicitly using **all**.

**select all** *bname*

**from** *account*

1. select \* means select all the attributes. Arithmetic operations can also be in the selection list.

**The where Clause**

1. The predicates can be more complicated, and can involve
   * Logical connectives **and, or** and **not**.
   * Arithmetic expressions on constant or tuple values.
   * The **between** operator for ranges of values.
2. Example: Find account number of accounts with balances between $90,000 and $100,000.

**select** *account#*

**from** *account*

**where** *balance* **between** *90000* **and** *100000*

## 

## The from Clause

1. The **from** class by itself defines a Cartesian product of the relations in the clause.
2. SQL does not have a natural join equivalent. However, natural join can be expressed in terms of a Cartesian product, selection, and projection.
3. For the relational algebra expression

displaymath1738

we can write in SQL,

**select distinct** *cname, borrower.loan#*

**from** *borrower, loan*

**where** *borrower.loan# = loan.loan#*

1. More selections with join: ``Find the names and loan numbers of all customers who have a loan at the SFU branch,'' we can write in SQL,

**select distinct** *cname, borrower.loan#*

**from** *borrower, loan*

**where** *borrower.loan# = loan.loan#*

**and** *bname=``SFU''*

**The Rename Operation**

1. Rename: a mechanism to rename both relations and attributes.
2. **as**-clause can appear in both the select and from clauses:

*old-name* **as** *new-name*.

1. Example.

**select distinct** *cname, borrower.loan#* **as** *loan\_id*

**from** *borrower, loan*

**where** *borrower.loan#* = *loan.loan#*

**and** *bname= ``SFU"*

**Tuple Variables**

1. Tuple variables can be used in SQL, and are defined in the **from** clause:

**select distinct** *cname, T.loan#*

**from** *borrower* ***as*** *S, loan* ***as*** *T*

**where** *S.loan# = T.loan#*

Note: The keyword **as** is optional here.

1. These variables can then be used throughout the expression. Think of it as being something like the rename operator.

Finds the names of all branches that have assets greater than at least one branch located in Burnaby.

**select distinct** *T.bname*

**from** *branch S, branch T*

**where** *S.bcity=``Burnaby''* **and** *T.assets > S.assets*

## String Operations

1. The most commonly used operation on strings is pattern matching using the operator **like**.
2. String matching operators **%** (any substring) and **\_** (underscore, matching any character).

E.g., ``\_\_\_%'' matches any string with at least 3 characters.

1. Patterns are case sensitive, e.g., ``Jim" does not match ``jim".
2. Use the keyword **escape** to define the *escape* character.

E.g., like ``ab%tely tex2html_wrap_inline1742 % tex2html_wrap_inline1742 '' escape `` tex2html_wrap_inline1742 '' matches all the strings beginning with ``ab'' followed by a sequence of characters and then ``tely'' and then ``% tex2html_wrap_inline1742 ''.

Backslash overrides the special meaning of these symbols.

1. We can use **not like** for string mismatching.
2. Example. Find all customers whose street includes the substring ``Main''.

**select** *cname*

**from** *customer*

**where** *street* **like** *``%Main%''*

1. SQL also permits a variety of functions on character strings, such as concatenating (using `` tex2html_wrap_inline1750 ''), extracting substrings, finding the length of strings, converting between upper case and lower case, and so on.

**Ordering the Display of Tuples**

1. SQL allows the user to control the order in which tuples are displayed.
   * **order by** makes tuples appear in sorted order (ascending order by default).
   * **desc** specifies descending order.
   * **asc** specifies ascending order.

**select** \*

**from** *loan*

**order by** *amount* **desc**, *loan#* **asc**

Sorting can be costly, and should only be done when needed.

**Duplicate Tuples**

* Formal query languages are based on mathematical relations. Thus no duplicates appear in relations.
* As duplicate removal is expensive, SQL allows duplicates.
* To remove duplicates, we use the **distinct** keyword.
* To ensure that duplicates are not removed, we use the **all** keyword.
* *Multiset* (bag) versions of relational algebra operators.
  + if there are tex2html_wrap_inline1754 copies of tuples tex2html_wrap_inline1756 in tex2html_wrap_inline1758 , and tex2html_wrap_inline1756 satisfies selection tex2html_wrap_inline1762 , then there are tex2html_wrap_inline1754 copies of tex2html_wrap_inline1756 in tex2html_wrap_inline1768 .
  + for each copy of tuple tex2html_wrap_inline1756 in tex2html_wrap_inline1758 , there is a copy of tuple tex2html_wrap_inline1774 in tex2html_wrap_inline1776 .
  + if there are tex2html_wrap_inline1754 copies of tuple tex2html_wrap_inline1756 in tex2html_wrap_inline1758 , and tex2html_wrap_inline1784 copies of tuple tex2html_wrap_inline1786 in tex2html_wrap_inline1788 , there is tex2html_wrap_inline1790 copies of tuple tex2html_wrap_inline1792 in tex2html_wrap_inline1794 .
* An SQL query of the form

**select** tex2html_wrap_inline1726

**from** tex2html_wrap_inline1728

**where** *P*

is equivalent to the algebra expression

displaymath1724

using the multiset versions of the relational operators tex2html_wrap_inline1802 , and tex2html_wrap_inline1804 .

REFERENCES  
Simon Fraser University, Burnaby, B.C., Canada