SQL

DATABASE MANAGEMENT SYSTEM

SUJAN TAMRAKAR

SQL (Structured Query Language)

- Comprehensive db language
- Contains queries for data definitions, updates, managements, view control, function and procedure controls.
- SQL => DDL, DML, DCL
  - DDL = provides commands for defining relation schemas, deleting relations, modifying relations schemas.
  - DML = includes a query language based on both the relational algebra and relational calculus. It provides commands to insert tuples, delete tuples, modify tuples.
  - DCL = used to control access to data stored in a database, relations, views (Authorization).
- Specifies Integrity Constraints that data must follow
- Includes commands for specifying the beginning & ending of transactions.

SQL (Structured Query Language)

- Embedded & Dynamic SQL : defines how SQL statements can be embedded with general purpose languages like C, C++, Java.

- Basic Domain Types:
  - Char(n): fixed length character string with user specified length ‘n’
  - Varchar(n): variable length character string with user specified maximum length ‘n’
  - Int, Small int, Real, float(n)
  - Date(yyyy-mm-dd)
  - Time(hh:mm:ss)
  - Timestamp – combination of date & time
  - Numeric(p,d): fixed point number with user specified precision (p= total digits, d=decimal eg. numeric(3,1) = 44.5)

MySQL DATA TYPES

DATE TYPE	SPEC	DATA TYPE	SPEC
CHAR	String (0 - 255)	INT	Integer (-2147483648 to 214748-3647)
VARCHAR	String (0 - 255)	BIGINT	Integer (-9223372036854775808 to 9223372036854775807)
TINYTEXT	String (0 - 255)	FLOAT	Decimal (precise to 23 digits)
TEXT	String (0 - 65535)	DOUBLE	Decimal (24 to 53 digits)
BLOB	String (0 - 65535)	DECIMAL	"DOUBLE" stored as string
MEDIUMTEXT	String (0 - 16777215)	DATE	YYYY-MM-DD
MEDIUMBLOB	String (0 - 16777215)	DATETIME	YYYY-MM-DD HH:MM:SS
LONGTEXT	String (0 - 4294967295)	TIMESTAMP	YYYYMMDDHHMMSS
LONGBLOB	String (0 - 4294967295)	TIME	HH:MM:SS
TINYINT	Integer (-128 to 127)	ENUM	One of preset options
SMALLINT	Integer (-32768 to 32767)	SET	Selection of preset options
MEDIUMINT	Integer (-8388608 to 8388607)	BOOLEAN	TINYINT(1)

SQL (Structured Query Language) Queries

- **CREATE** (to create either database or table)

  Syntax:

  Create table r (A_1 D_1, A_2 D_2, …, A_n D_n), (integrity $constraint_1$), … (integrity constraint $_n$);

  r = name of relation, A= attribute names in relation r, D= data type of attribute A,

  Integrity constraint= Primary Key, Foreign Key, Unique Key

  Example:

  Create database school;

  Create table customer (

  customer_name char(20), customer_street varchar(30), customer_city nvarchar(20), primary key (customer_name)

  );

  Primary are required to be non null & unique. It can consist of either one attribute or set of attributes.

SQL (Structured Query Language) Queries

- **INSERT** (to load data into the relation)

  Syntax:

  Insert into <tablename> values (value1, value2, value3… value n);

  Insert into <tablename> (attribute1, attribute2, … attribute n) values(value1, value2,…value n);

  Example:

  1. Insert into account values (‘A101’, ‘Pokhara’, 500);
  2. Insert into account (acct_no, branch, balance) values (‘A101’, ‘Pokhara’, 500);
  3. Insert into account (branch, acct_no, balance) values (‘Pokhara’, ‘A101’, 500);
  4. Insert into account (select loan_no, branch, 200 from loan where branch = ‘Pokhara’);

In last nested query, sub-query (child query) is executed first and then the parent query.

SQL (Structured Query Language) Queries

- **DELETE** (to delete tuples from a relation)

  Syntax:

  Delete from <tablename>;

  Delete from <tablename> where (condition);

  Example:

  1. Delete from loan;

  2. Delete from account where branch = ‘Pokhara’;

  3. Delete from loan where amount between 1300 and 2500;

  4. Delete from account where branch in (select branch_name from bank_branches where branch_city = ‘Pokhara’); ⇒ deletes all tuples at every branch located in Pokhara.

  5. Delete from account where balance < (select avg(balance) from account);

SQL (Structured Query Language) Queries

- **UPDATE** (to edit tuple(s) in a relation)

  Syntax:

  Update <table_name> set <column_name> = <value> where <condition>;

  Example:

  1. Update student set age = 18 where s_id = 102;
  2. Update student set s_name = ‘ram’, age = 17 where s_id = 103;
  3. Update student set attendance = ‘present’;

  Be careful when updating records. If we omit the WHERE clause, then all records column will be updated as per the query.

SQL (Structured Query Language) Queries

- **SELECT** (to display attribute's values or records full of attribute values). Consists of 3 clauses i.e. Select, From, Where.

<u>Syntax:</u>

Select A1, A2, A3,… An from R1, R2,.. Rn where P; [P= predicate/condition]

<u>Example:</u>

1. Select * from loan;
2. Select branchName from loan;
3. Select distinct branchName from loan;
4. Select loanNo from loan where branchName= 'Pokhara' and amount > 1200;
5. Select loanNo from loan where amount between 9000 and 15000;
6. Select loanNo, branchName, amount*100 from loan; ⇒ amount will be shown 100x multiplied
7. Select cusName, borrower.loanNo, amount from borrower, loan where borrower.loanNo=loan.loanNo; ⇒ no mention of table defaults to that table which contains that attribute

SQL (Structured Query Language) Queries

- **RENAME & ALIAS** (renaming relations, aliasing attributes for instance display)

  Syntax:

  Rename table ‘oldTableName’ to ‘newTableName’;

  Select Tbl.Attribute as Attrib from Table as tbl;

  Example:

  Rename table studrnt1 to student;

  Select cusName, loanNo as loanId, amount from loan;

  Select regd_indiv_lic_no as driverLicense, veh_regd_owner as driverName from license;

  Select cusName, B.loanNo, L.amount from borrowser as B, loan as L where B.loanNo=L.loanNo;

SQL (Structured Query Language) Queries

- **String operations** (to operate on pattern matching with ‘like’ keyword, uses special characters % and _)

  <u>Example:</u>

  Pokh% = matches any string beginning with Pokh

  %idge% = matches any string containing with ‘idge’ as substring. Ex: perryridge, rockridge,…

  _ _ _ = matches any string of exactly 3 characters

  _ _ _ % = matches any string of at least 3 characters

  Select customerName from customer where customer_street like ‘%aur’; ⇒ Lamachaur, batulechaur, simalchaur, puranchaur…..

  Select * from Customers where CustomerName like '%or%'; ⇒ cory, dorthy, norah, …

  Other string operations: concatenating, extracting substrings, finding length, case conversion,..

SQL (Structured Query Language) Queries

- **Ordering** (need to be careful for larger number of tuples to avoid higher cost)
  - ‘order by’ clause causes the tuples in the result of a query to appear in sorted order either in ascending or descending order
  - By default, ‘order by’ works as ascending order
  - For explicitly mentioning the sort order, asc is used for ascending & desc for descending.

Select distinct customerName from borrower where branchName = ‘Pokhara’ order by customerName;

Select * from loan order by amount desc, loanNo asc;

(this last query orders the result in descending order based on amount but if multiple tuples have same amount then on those tuples records will be sorted in ascending order based on loanNo attribute)

SQL (Structured Query Language) Queries

- **Set operations** (union, intersect, except)

1. **Union –** to find all possible data meeting the criteria, operates on relations.
   - To find all bank customers having a loan, an account or both at the bank.
   - Ex: (Select cusname from depositor) union (select cusname from borrower) this operation automatically eliminates duplicates unlike select command.

     If a customer 'John' has several accounts or loan or both at the bank, then John will appear only once in the result.

   - If we want to retain all duplicates, we must write 'Union all' in place of union.
   - Ex: (Select cusname from depositor) union all (select cusname from borrower)

     If a customer 'John' has 3 accounts and 2 loans at the bank, then there will be 5 tuples of John in the result.

SQL (Structured Query Language) Queries

- **Set operations** (union, intersect, except)

2. **Intersect–** to find the common data meeting the criteria.

   - To find all bank customers having both loan and account at the bank.
   - Ex: (Select cusname from depositor) intersect (select cusname from borrower) this operation automatically eliminates duplicates.

     If a customer 'John' has several accounts and loan at the bank, then John will appear only once in the result.

   - If we want to retain all duplicates, we must write 'Intersect all' in place of Intersect.
   - Ex: (Select cusname from depositor) intersect all (select cusname from borrower)

     If a customer 'John' has 3 accounts and 2 loans at the bank, then there will be 2 tuples of John in the result.

SQL (Structured Query Language) Queries

- **Set operations** (union, intersect, except)

3. **Except–**to find the data present in one relation and not in other

   - To find all bank customers having an account but no loan at the bank.
   - Ex: (Select cusname from depositor) except (select cusname from borrower) this operation automatically eliminates duplicates.

     A tuple with customer John will appear (exactly once) in result only if John has an account at bank & no loan.

   - If we want to retain all duplicates, we must write 'Except all' in place of Except.
   - Ex: (Select cusname from depositor) except all (select cusname from borrower)

     If John has 3 accounts & 1 loan, then there will be 2 tuples with name John in result.

     But if John has 2 accounts & 3 loans, then there will be no tuple with names John in result.

SQL (Structured Query Language) Queries

- **Aggregate operations**
  - These operations take a collection of values as input and return a single value.
  - 5 built-in aggregate functions:
    - Min – finding smallest value from that attribute
    - Max – finding largest value from that attribute
    - Avg – finding average value from within that attribute domain
    - Sum – summer of particular attribute
    - Count – calculating total number of items

Example:

Select avg(balance) from account where branchName = ‘Pokhara’;

Select branchName, avg(balance) from account group by branchName;

[group by = tuples with same value on all attributes in such clause are placed in one group]

Thank You!