Structured Query Language

SQL Data Definition Language

Table Creation - Data Types

- Character Data - if you want to store strings, for example up to 50 characters in length, you could use:
  - char(50)   /* fixed-length */
  - varchar(50)   /* variable-length */
- DataBase dependent:
- in MySQL char columns is currently 255 chars, whereas varchar columns can be up to 65,535 chars
- in SQL Server there are also nchar(), nvarchar(), store Unicode characters (essential if you require use of extended character sets); 2 vs 4 bytes each char

Table Creation - Data Types

- Character Data - if you want to store longer strings (such as emails, documents, etc.), then you will want to use in MySQL one of text types (mediumtext and longtext), or in SQL Server varchar(MAX) or nvarchar(MAX)
- use *char* when all strings to be stored in column are of similar length, such as state abbreviations, phone numbers and *varchar* when strings to be stored in column are of varying lengths
- both *char* and *varchar* are used in similar fashion in all major database servers (engines)

Table Creation - Data Types

- Character sets: databases can store data using various character sets, both single/multibyte, some of supported character sets in MySQL

Charset	Description	Default collation	MaxLen (in bytes)
big5	Big5 Traditional Chinese	big5_chinese_ci	2
cp850	DOS West European	cp850_general_ci	1
hp8	HP West European	hp8_english_ci	1
koi8r	KOI8-R Relcom Russian	koi8r_general_ci	1
latin2	ISO 8859-2 Central European	latin2_general_ci	1
ascii	US ASCII	ascii_general_ci	1
ujis	EUC-JP Japanese	ujis_japanese_ci	3
hebrew	ISO 8859-8 Hebrew	hebrew_general_ci	1
gb2312	GB2312 Simplified Chinese	gb2312_chinese_ci	2
greek	ISO 8859-7 Greek	greek_general_ci	1
gbk	GBK Simplified Chinese	gbk_chinese_ci	2
armscii8	ARMSCII-8 Armenian	armscii8_general_ci	1
utf8	**UTF-8 Unicode**	**utf8_general_ci**	**3**

Table Creation - Data Types

- Character Data - to work with string ranges, you need to know order of characters within your character set (order in which characters within character set are sorted is called a ***collation***).

Table Creation - Data Types

- Character Data
- if you create column for free-form data entry, such as notes column to hold data about customer interactions with your company's customer service department, then *varchar* will probably be adequate
- if you are storing documents, however, you should choose *longtext* type

Table Creation - Data Types

- Numeric Data - there are several different numeric data types that reflect various ways in which numbers are used
- column indicating whether customer order has been shipped - referred to as Boolean, would contain 0 to indicate false and 1 to indicate true
- system-generated primary key for transaction table - generally start at 1 and increase in increments of 1 up to potentially very large number

Table Creation - Data Types

- Numeric Data
- item number for customer’s electronic shopping basket - values for this would be positive whole numbers between 1 and, at most, 200
- price for item from customer’s electronic shopping basket - values for this would have 2 decimal positions (money)
- positional data for circuit board drill machine - high-precision scientific or manufacturing data often requires accuracy to 8 decimal points

Table Creation - Data Types

- Numeric Data
- *int* in MySQL, *integer* in SQL Server represent numbers between −2,147,483,648 to 2,147,483,647ș or from 0 to 4,294,967,295 for unsigned variant
- *Float*(p,s) −3.402823466E+38 to −1.175494351E-38 and 1.175494351E-38 to 3.402823466E+38
- *Double*(p,s) −1.7976931348623157E+308 to −2.2250738585072014E-308 and 2.2250738585072014E-308 to 1.7976931348623157E+308

- using floating-point type, you can specify *precision* (total number of allowable digits both to left and to right of decimal point) and *scale* (number of allowable digits to right of decimal point), but they are not required
- data stored in column will be rounded if number of digits exceeds scale and/or precision of column
- for example, column defined float(4,2) will store total of four digits, two to left of decimal and two to right of decimal; number 17.8675 would be rounded to 17.87, and attempting to store number 178.375 would generate error

Table Creation - Data Types

- Temporal Data
- for character strings and numerical data there are pretty much the same way of storing values and same function in all (relational) databases: MySQL, SQL Server, Access, Oracle, DB2, SQLite, PostgreSQL, …
- different ways of storing temporal data in databases

MySQL temporal types

- *Type* Default format Allowable values
- *Date* YYYY-MM-DD 1000-01-01 to 9999-12-31
- *Time* HHH:MI:SS –838:59:59 to 838:59:59
- *Datetime* YYYY-MM-DD HH:MI:SS 1000-01-01 00:00:00 to 9999-12-31 23:59:59
- *Timestamp* YYYY-MM-DD HH:MI:SS 1970-01-01 00:00:00 to 2037-12-31 23:59:59
- *Year* YYYY 1901 to 2155

SQL SERVER temporal types

- Data type Format Range Accuracy Storage size (bytes)
- *Time* hh:mm:ss[.nnnnnnn] 00:00:00.0000000 through 23:59:59.9999999 100 nanoseconds 3 to 5
- *Date* YYYY-MM-DD 0001-01-01 through 9999-12-31 1 day 3
- *Smalldatetime* YYYY-MM-DD hh:mm:ss 1900-01-01 through 2079-06-06 1 minute 4
- *Datetime* YYYY-MM-DD hh:mm:ss[.nnn] 1753-01-01 through 9999-12-31 0.00333 second 8
- *Datetime2* YYYY-MM-DD hh:mm:ss[.nnnnnnn] 0001-01-01 00:00:00.0000000 through 9999-12-31 23:59:59.9999999 100 nanoseconds 6 to 8
- *Datetimeoffset* YYYY-MM-DD hh:mm:ss[.nnnnnnn] [+|-]hh:mm 0001-01-01 00:00:00.0000000 through 9999-12-31 23:59:59.9999999 (in UTC) 100 nanoseconds 8 to 10 (TimeZone OffSet)

Table Creation

- CREATE TABLE person
- (person_id SMALLINT UNSIGNED,
- firstname VARCHAR(20),
- lastname VARCHAR(20),
- gender CHAR(1),
- birth_date DATE,
- street VARCHAR(30),
- city VARCHAR(20),
- state VARCHAR(20),
- country VARCHAR(20),
- postal_code VARCHAR(20),
- CONSTRAINT pk_person PRIMARY KEY (person_id));
- *Query OK, 0 rows affected*

- everything should be fairly self-explanatory …
- except for last item - when define table, need to tell database server what column or columns will serve as Primary Key - do this by creating *constraint* on table (can add several types of constraints to table definition)
- CONSTRAINT pk_person PRIMARY KEY (person_id) - created on person_id column and given name pk_person and is of type PK

- another type of constraint called *check constraint* constrains allowable values for particular column
- MySQL
  - gender CHAR(1) CHECK (gender IN ('M','F')),
- SQL Server
  - CONSTRAINT [CK_Products_UnitPrice] CHECK (([UnitPrice] >= 0))

Table Creation

- if you forget to create constraints when you first create table, you can add it later via
- ALTER TABLE table_name
- tables which will not be used could be eliminated form database schema by issuing
- DROP TABLE table_name

Establishing Relationships between Tables

- relationships are established by columns with values that are shared/match between two tables
- in a one-to-many relationship, one row (from one column) in first table matches same value in multiple rows in one column of second table
- database that is properly linked together using Foreign Keys is said to have *referential integrity*

Establishing Relationships between Tables

- ALTER TABLE Products WITH NOCHECK ADD CONSTRAINT FK_Products_Categories FOREIGN KEY(CategoryID) REFERENCES Categories (CategoryID)
- adding Foreign Key constraints to table on many side of one-to-many relationship creates links
- Foreign Key constraints need to be added to only one side of relationship, in a one-to-many relationship, they are added to many side

Database Schema creation

- usually these operations are done in front-end tools like MySQL Workbench or SQL Server Management Studio
- which translate DataBase Addministrator intent into SQL language (proper CREATE, ALTER, DROP tables)
- everything which is done in database server / engine it is actually SQL language

SQL Data Manipulation Language

CRUD CREATE (INSERT) READ (SELECT) UPDATE DELETE

Codd’s rules (for relational DataBase)

- 7. **The system must support set-at-a-time insert, update, and delete operations.**
- means that system must be able to perform insertions, updates, and deletions of multiple rows in single operation

- simplest way to populate character column is to enclose string in quotes
- numeric data is quite straightforward
- working with temporal data is most involved when it comes to data generation and manipulation; some of complexity is caused by many ways in which single date and time can be described
- Wednesday, September 15, 20010
- 9/15/2010 2:14:56 P.M. EST 9/15/2010 19:14:56 GMT
- 2612008 (Julian format) Star date [−4] 85712.03 14:14:56 (*Star Trek* format)
- and you have to deal also with Time Zones

- INSERT INTO string_tbl (char_fld, vchar_fld, text_fld) VALUES ('This is char data', 'This is varchar data’, 'This is text data');
- *Query OK, 1 row affected*
- UPDATE string_tbl SET vchar_fld = 'This is a piece of extremely long, too long varchar data';
- *ERROR : Data too long for column 'vchar_fld' at row 1*
- DELETE FROM string_tbl;
- *Query OK, 1 row affected (0.00 sec)*

including ' single quotes

- won’t be able to insert following string because server will think that apostrophe in word *doesn’t* marks end of string
- UPDATE string_tbl SET text_fld = 'This string doesn't work';
- need to add *escape* to string so that server treats apostrophe like any other character in string.
- all servers allow you to escape single quote by adding another single quote directly before:
- UPDATE string_tbl SET text_fld =
- 'This string didn''t work, but it does now';
- *Query OK, 1 row affected (0.01 sec)*
- *Rows matched: 1 Changed: 1 Warnings: 0*

- UPDATE Orders SET OrderDate = '2016-07-04'
- WHERE OrderID = '10248’
- INSERT INTO Customers (
- CustomerID, CompanyName, ContactName, ContactTitle, Address, City, Region, PostalCode, Country, Phone) VALUES
- ('UTC-N’, 'Technical University of Cluj-Napoca’, 'Calin CENAN’, 'Lecturer’, 'Baritiu St. 26-28’, 'Cluj-Napoca’, 'Cluj’, '400124’, 'Romania’, '0264 401 200’);

Adding incomplete records

- sometimes you might want to add a record to table before you have data for all record’s columns – possible as long as you have data for primary key and all columns that have NOT NULL constraint
- INSERT INTO Customers VALUES
- ('UTC-N’, 'Technical University of Cluj-Napoca’, 'Calin CENAN’, 'Lecturer’, 'Baritiu St. 26-28’, 'Cluj-Napoca’, 'Cluj’, '400124’, 'Romania’, '0264 401 200’, NULL); value of last column, Fax, is NULL

- INSERT INTO Products (ProductName, SupplierID, CategoryID, QuantityPerUnit)
- VALUES ('Tirigomodina’, 1, 1, ‘’)
- ProductID is automatically inserted by DB engine as next value in sequence
- UnitPrice, UnitsInStock, UnitsOnOrder, ReorderLevel, Discontinued have default value of 0

- SELECT * INTO Shipped FROM Orders
- WHERE 0=1;
- copies data from one table into new table (without any constraints or keys from original table)
- copy all columns into new table and no rows (condition it is always false)
- create new, empty table using schema of another
  – just add WHERE clause that causes query to return no data

- INSERT INTO Shipped (CustomerID, EmployeeID, OrderDate,[RequiredDate, ShippedDate, ShipVia, Freight, ShipName, ShipAddress, ShipCity, ShipRegion, ShipPostalCode, ShipCountry)
- SELECT CustomerID, EmployeeID, OrderDate,[RequiredDate, ShippedDate, ShipVia, Freight, ShipName, ShipAddress, ShipCity, ShipRegion, ShipPostalCode, ShipCountry FROM Orders WHERE Orders.ShippedDate IS NOT NULL;

- keying in a succession of SQL INSERT statements is the slowest and most tedious way to enter data into database
- although clerks at airline ticket counters, fast food restaurants, and supermarkets and users at e-commerce Web sites don’t need to know anything about SQL, somebody does
- in order to make users’ life easier, someone has to write programs that process data coming in from keyboards, data entry pads, and bar code scanners, and sends it to database
- those programs are typically written in general-purpose language such as C, Java, Visual Basic or php and incorporate SQL statements that are then used in actual ‘conversation’ with database

Updating data in table

- UPDATE *table_name* SET *column_1 = expression_1, column_2 = expression_2, ..., column_n = expression_n*
- [WHERE predicates] ;

- UPDATE Orders SET
- OrderDate = DATEFROMPARTS (2017, MONTH(OrderDate), DAY(OrderDate)) ,
- RequiredDate = DATEFROMPARTS (2017, MONTH(RequiredDate), DAY(RequiredDate)) ,
- ShippedDate = DATEFROMPARTS (2017, MONTH(ShippedDate), DAY(ShippedDate))
- *(830 rows affected)*

- SET clause specifies which columns will get new values and what those new values will be
- optional WHERE clause specifies which rows update applies to
  - if there is no WHERE clause, update is applied to all rows in table
- UPDATE Products SET ProductName = 'Ce Ai?!'
- WHERE ProductName = 'Chai’
- WHERE ProductID = 1

Deleting data from table

- DELETE FROM *table_name*
- [WHERE predicates] ;
- optional WHERE clause specifies which rows delete applies to
  - if there is no WHERE clause, delete is applied to all rows in table
- DELETE FROM Orders
- WHERE OrderID IN
- (SELECT OrderID FROM Shipped)

- DELETE FROM Orders
- WHERE OrderDate < ‘2017-01-01’
- all of 2016’s Orders records (and previously) are deleted

SQL Data Control Language

SECURITY

- GRANT and REVOKE control who may access various parts of database
- before you can grant database access to someone, you must have some way of identifying that person - some parts of user identification, such as issuing passwords and taking other security measures, are implementation-specific
- SQL has a standard way of identifying and categorizing users, however, so granting and revoking privileges can be handled

- SQL doesn’t specify how user identifier is assigned; in many cases, operating system’s login ID serves purpose
- role name it is other form of authorization identifier that enable access to a database
- every SQL session is started by user and have a *SQL-session user identifier*
- privileges associated with SQL-session user identifier determine what privileges that user has and what actions she may perform

- in small company, identifying users individually doesn’t present any problem
- in larger organization roles come in - although company may have large number of employees, these employees do a limited number of jobs; all sales clerks require same privileges, all warehouse workers require different privileges - they play different roles in company
- job of maintaining authorizations for everyone is made much simpler
- new user is added to SALES_CLERK role name and immediately gains privileges assigned to that role; sales clerk leaving company is deleted from role; employee changing from one job category to another is deleted from one role name and added to another
- just as session initiated by a user is associated with SQL-session user identifier, it is also associated with SQL-session role name

Roles

- create role with :
- CREATE ROLE <role name>
- [WITH ADMIN {CURRENT_USER | CURRENT_ROLE}] ;
- when create role, role is automatically granted to you; also granted right to pass role-creation privilege on to others
- for destroying role:
- DROP ROLE <role name> ;

Users classes of users

1. **DataBase Administrator (DBA):** responsibility of DBA is to maintain database; have full rights to all objects in database; can also decide what privileges other users may have
2. **Database object owners:** users who create database or database objects such as tables and views are automatically owners of those objects; possesses all privileges related to object; database object owner's privileges are equal to those of DBA, but only with respect to object in question
3. **Grantees:** are users who have been granted selected privileges by DBA or database object owner; may or may not be given right to grant her privileges to others
4. **public:** all users are considered to be part of public, regardless of whether they have been specifically granted any privileges; privileges that are granted to PUBLIC may be exercised by any user

Granting Privileges

- GRANT <privilege list>
- ON <privilege object>
- TO <user list> [WITH GRANT OPTION]
- [GRANTED BY {CURRENT_USER | CURRENT_ROLE}] ;
- <privilege list> ::= privilege [, privilege]...

- <privilege> ::=
- SELECT [(<column name> [, <column name>]...)]
- | SELECT (<method designator> [, <method designator]...)
- | DELETE
- | INSERT [(<column name> [, <column name>]...)]
- | UPDATE [(<column name> [, <column name>]...)]
- | REFERENCES [(<column name> [, <column name>]...)]
- | USAGE
- | TRIGGER
- | UNDER
- | EXECUTE

- <privilege object> ::=
- [TABLE] <table name>
- | <view name>
- | DOMAIN <domain name>
- | CHARACTER SET <character set name>
- | COLLATION <collation name>
- | TRANSLATION <translation name>
- | TYPE <user-defined type name>
- | <specific routine designator>

- <user list> ::=
- authorizationID [, authorizationID]...
- | PUBLIC

- a lot of syntax …
- not all privileges apply to all privilege objects
- SELECT, DELETE, INSERT, UPDATE, and REFERENCES privileges apply to TABLE
- SELECT privilege also applies to VIEWS
- USAGE privilege applies to DOMAIN, CHARACTER SET, COLLATION, and TRANSLATION
- TRIGGER privilege applies to TRIGGER
- UNDER privilege applies to user-defined types
- EXECUTE privilege applies to specific routines

- GRANT SELECT
- ON CUSTOMER
- TO SALES_MANAGER ;
- enables sales manager to query CUSTOMER table
- GRANT UPDATE
- ON CUSTOMER
- TO SALES_MANAGER ;
- enables the sales manager to change data

- GRANT ALL PRIVILEGES
- ON sinu
- TO rector;
- doing it all; for highly trusted person who has just been given major responsibility, rather than issuing a series

- GRANT UPDATE
- ON CUSTOMER
- TO SALES_MANAGER WITH GRANT OPTION ;
- passing on the power; similar to GRANT UPDATE example but also gives the right to grant update privilege to anyone she/he wants

Revoking Privileges

- REVOKE [GRANT OPTION FOR] <privilege list>
- ON <privilege object>
- FROM <user list>
- [GRANTED BY {CURRENT_USER | CURRENT_ROLE}]
- {RESTRICT | CASCADE} ;

- major difference from GRANT syntax is addition of
- RESTRICT and CASCADE keywords and it isn’t optional - one of two keywords is required
- SQL Server CASCADE keyword is optional, and RESTRICT sense is default assumed if CASCADE is not present
- if includes RESTRICT keyword, checks to see whether privilege being revoked was passed on to one or more other users; if it was, privilege isn’t revoked, and you receive error message
- if includes CASCADE keyword, revokes privilege, as well as any dependent instances of this privilege that were granted by instance you’re revoking

- with optional GRANT OPTION FOR clause, you can revoke user’s ability to grant privilege without revoking his ability to use privilege

Granting Roles

- GRANT <role name> [{ , <role name>}...]
- TO <user list>
- [WITH ADMIN OPTION]
- [GRANTED BY {CURRENT_USER | CURRENT_ROLE}] ;
- can grant any number of roles to names in list of users
- if you want to grant role and extend to grantee right to grant same role to others, you do so with the WITH ADMIN OPTION clause

Revoking Roles

- REVOKE [ADMIN OPTION FOR] <role name> [{ , <role name>}...]
- FROM <user list>
- [GRANTED BY {CURRENT_USER | CURRENT_ROLE}]
- {RESTRICT | CASCADE}
- revoke one or more roles from users in user list
- can revoke admin option from role without revoking role itself

- as always it is not possible to close whiteout **thank you for your kindly attention!**
- *If you get half as much pleasure - the guilty variety, to be sure - from reading this slides as I get from writing it, we're all doing pretty well.*