SQL

Sorry Trent, no more America's Next Top Model

What is it?

- Structured Query Language
- Meant for use with relational databases
  - Its parent is SEQUEL from IBM
  - Based on Relational Algebra if you're a math nerd
- Structured **Query** Language
  - Hmmm..... what do you think it is used for?

What's it for?

- It is a **declarative** language
  - Program *flow* is not expressed, just the *logic*
  - Basically you declare what you want and the details of figuring it out are left to the language and system
- Thus, the basic idea is you will be asking a lot of queries, or questions and getting answers
- Additionally...
  - Creation and structure
  - Data manipulation

Oops one problem

- ANSI has a SQL standard just like C
- Unfortunately there are different dialects of SQL
  - Each database system may interpret some commands slightly differently or offer their own non-standard commands
  - Example: boolean type in the standard is *optional* (how dumb is that) so it may or may not exist in your database
- What does this mean? Well you'll often learn your own 'accent' of SQL depending on which system you typically use

The two (three?) pieces

- SQL has two jobs
  - **Data definition** → how you set up the data structure
  - **Data manipulation** → how you deal with the data
- Well really there's three, the querying ability, we'll get to that later

Data Definition

- This is when the database structure is defined
  - Before beginning, you want to have an ERD created so you know what you're defining!
- Step 1: Create a database (duh)
  - Basically your RDBMS handles this
  - Physical file creation

Aside: Authentication & Permissions

- RDBMSes have users and you must be logged in to use them
- Users have particular permissions and views of the databases and their data
- For instance, you must have DB creation permissions to even get through step 1

Data Definition – Schemas

- Schemas are not quite standard and the term is used for slightly different things at different times
- Typically, a schema is a view of the parts of a database
  - For example, the administrator of a DB probably can see everything whereas the accountants can only see the pieces that relate to their work
- MS SQL supports these but we won't worry about implementing them

Before we begin

We'll follow along with the book figures so let's look at Fig 7.1 and 7.2 as well as the notes following 7.2 about the data

Table Creation (Step 2 of DD)

- For this we use a command called ***drum roll*** **...** CREATE TABLE
- Let's examine the syntax of the command

CREATE TABLE

- Command is followed by the name of the table you wish
- Cannot have spaces nor mathematical symbols, best to avoid special characters entirely other than _

```
CREATE TABLE tablename
(
) ;
```

CREATE TABLE

- Parenthesis surround the entirely command
- Note the semicolon at the end

```
CREATE TABLE tablename
(
) ;
```

CREATE TABLE

- Columns of the table are specified by name, data type, and if they have any constraints

CREATE TABLE *tablename*

(

col1 *datatype [constraint]*[,

col2 *datatype [constraint]*][,

PRIMARY KEY (**colw**[, **colx**])][,

FOREIGN KEY (**coly**[, **colz**])

REFERENCES **table**][,

CONSTRAINT **constraint**]

) ;

CREATE TABLE

- How you denote the PK, list multiple columns separated by commas
- Quick way to do single column PKs...

CREATE TABLE *tablename*

(

col1 *datatype* [*constraint*][,

col2 *datatype* [*constraint*]][,

PRIMARY KEY (**colw**[, **colx**])][,

FOREIGN KEY (**coly**[, **colz**])

REFERENCES **table**][,

CONSTRAINT **constraint**]

) ;

CREATE TABLE

- How you denote the FK, list multiple columns separated by commas
- Quick way to do single column Fks...
- NOTE! The **constraints** part is new on this slide

CREATE TABLE *tablename*

(

col1 *datatype* [*constraint*][,

col2 *datatype* [*constraint*]][,

PRIMARY KEY (**colw**[, **colx**])][,

FOREIGN KEY (**coly**[, **colz**])

REFERENCES **table**] **[constraints]**[,

CONSTRAINT **constraint**]

) ;

CREATE TABLE

- Last but certainly not least
- These will be mostly the ***check*** or ***unique*** variety
- Note that PK and FK definitions are also considered constraints

CREATE TABLE *tablename*

(

col1 *datatype* [*constraint*][,

col2 *datatype* [*constraint*]][,

PRIMARY KEY (**colw**[, **colx**])][,

FOREIGN KEY (**coly**[, **colz**])

REFERENCES **table**] [**constraints**][,

CONSTRAINT **constraint**]

) ;

Data Types

- We have touched on these before but let's examine them
- Very similar to the way we define types in programming languages, but of course it has its own rules and syntax
- Pretty good reference: http://webcoder.info/reference/MSSQLDataTypes.html

Numbers

- bigint, int, smallint, tinyint
- bit → 0 or 1, basically stands in for boolean
- decimal[(p[, s])] → a decimal number with precision p (1 to 38, default 18) and scale s (the number of decimal places, default 0, 0<=s<=p)
  - numeric is the same thing
- float → decimal numbers with floating decimal places
  - real → a float with less digits

Sort-of Numbers

- money, smallmoney → odd thing here is they store 4 decimal places, what else could you use?
- datetime → wide range of dates, from 1753 through 9999
- smalldatetime → only 1900 through 2079

Characters and Text

- Char[(n)] and varchar[(n)] → char is fixed length and varchar is variable length strings, up to 8000 characters (specified by n, default is 1)
  - What situations are appropriate for each?
- Text → like varchar but up to 2^31-1 characters
- All of these are non-Unicode, to support Unicode add an 'n' on the front of each type → will have to use these in SQL Server 2008

Binary types

- binary[(n)] and varbinary[(n)] → any binary data up to 8000 bytes of storage
- image → up to 2^31-1 byes of storage

Figuring out data types

- Let's look at Fig. 7.2 and see what we think

Data Constraints

- NOT NULL → doesn't allow a column to be null
- UNIQUE → only unique values, repeats generate errors when inserting
  - Can be specified with a column or in the CONSTRAINTs at the bottom, the latter way is the only way to mark more than one column unique
  - Marking something as UNIQUE automatically creates an index for it
- DEFAULT → what a column gets if not specified
  - For dates GETDATE() is helpful

Creating PKs and FKs

- PK quick way for a single column:
  - column_name [constraints] PRIMARY KEY
- Better way (seen in the create table slides)
  - PRIMARY KEY (col1[, col2...])
- FK quick way for a single column
  - column_name [constraints] REFERENCES table
- Better way
  - FOREIGN KEY (col1[, col2...]) REFERENCES table (refcol1[, refcol2...]) [constraints]

PK Constraints

- All columns involved in a PK should be UNIQUE and NOT NULL
- For cases where you want a surrogate or "automatic" key, these should have IDENTITY(start, increment) right after the data type

FK Constraints

- Referential integrity is automatically enforced
- Can only reference columns marked UNIQUE or PRIMARY KEY
- ON UPDATE CASCADE → important!
  - Goes after "REFERENCES *tablename*"
- ON DELETE CASCADE
  - May or may not be good, depending on situation

CHECK Constraints

- Used for data validation
- Can be any expressions that evaluate to a boolean
- Ex. **CONSTRAINT** GPA_CHK **CHECK** (GPA >= 0 AND GPA <= 4)
- Note that check constraints can also be listed in the data constraints for a column, in which case you just go from CHECK onward, you don't give it a name

CHECK Examples

- Only area codes 513 or 859?
- Only valid prices?
- Discount not over 20%?
- Dates past the company founding date (we'll use today)?

CHECK answers

- CONSTRAINT ac_chk CHECK V_AREACODE = 859 OR V_AREACODE = 513
  - CONSTRAINT ac_chk CHECK V_AREACODE IN (859, 513)
- CONSTRAINT prc_chk CHECK P_PRICE > 0
- CONSTRAINT dsc_chk CHECK P_DISCOUNT > 0 AND P_DISCOUNT < .2
  - CONSTRAINT dsck_chk CHECK P_DISCOUNT BETWEEN 0 AND .2
- CONSTRAINT date_chk CHECK CONVERT(CHAR(10),P_INDATE,120) > '2010-04-25')

Warning: Lazy Professor Ahead

Indexes

- Book index → good analogy
- ***Index key*** → the attributes that an index is based on
- ***Unique Index*** → an index that has only one row associated with it
  - Primary keys are common index keys for these
  - UNIQUE constraints create these

Indexes

- Typically used for fast lookup
- ***Covering Index*** → a special index that contains all the attributes needed for a query that will take advantage of it of the index
  - Can be very costly!
- When trying to find good indexes, think about what the attributes you're typically going to search on/for

Indexes

- Speed up read queries *a lot*
- Slow down writes
- Take up extra space, especially covering indexes
- Cannot help with some types of wildcard queries
- A table can have many indexes, make use of this fact!

SQL Indexes

- Enough with the theory!
- As mentioned, UNIQUE creates them automagically
- Syntax →
  CREATE [UNIQUE] INDEX *indexname* ON
  tablename(**col1**[, **col2**...])
  DROP INDEX *indexname*
- If we wanted to create a report of all products from a vendor what index would we create and what would be the command? Should it be marked unique?

Aside: unique?

- UNIQUE data constraint is one column
- UNIQUE table constraint is one or more columns
- Both create a unique index automatically
- Columns marked unique can be used as FKs
- A unique index is roughly the same thing but does not give you FK ability

Data Manipulation

- These commands allow you to enter, change, and delete information as well as undo changes
- INSERT
- UPDATE
- DELETE
- COMMIT
- ROLLBACK

Insert

- A way of entering new rows
- Syntax (simple) →
  INSERT INTO *table* VALUES (val1,
  val2, ...)
- Have to watch out for FK referential integrity violations here
- All columns must have a value given in the VALUES list
  - If a column allows nulls, you can use NULL for it if no data is available

Insert

- But what if I only have data for certain columns?
- Syntax →
  INSERT INTO table (col1, col2,...) VALUES (val1, val2, ...)
- Important: the column names and values must be in the same order!

Values for Inserts (and others)

- Characters, strings, and dates are all entered between single quotes → '
- Numbers aren't (otherwise they'd be strings...)
- Constraint keywords are not separated by commas, but values and columns are
- In the first insert form showed, you have to enter values for all columns

SELECT: A Preview

- The way to access information is the SELECT command
- Simple Syntax →
  SELECT columnlist FROM table
- Get all rows in a table →
  SELECT * FROM table
- * is a wildcard... remember these?

Back to INSERT

- You can use INSERT in combination with SELECT
- Syntax →
  INSERT INTO table [(col1, col2...)]
  SELECT columnlist FROM table
- The columnlist in SELECT does not have to have the same names as the columns, but the data types must match
- The SELECT is a query in and of itself, so this is called using a nested query

UPDATE

- Whoops, I made a mistake.... how do I fix it?
- Syntax →
  UPDATE table
  SET column = expression[, col=exp...]
  [WHERE conditionlist]
- Where has two typical uses:
  - Identifying a particular column
  - Identifying a range of columns based on an expression
- Update is not often used without a where clause

DELETE

- Syntax →
  DELETE FROM table
  [WHERE conditionlist]
- Recall referential integrity and cascading deletes from FKs
- Careful! A delete without a where clause can be disastrous

A couple tricks

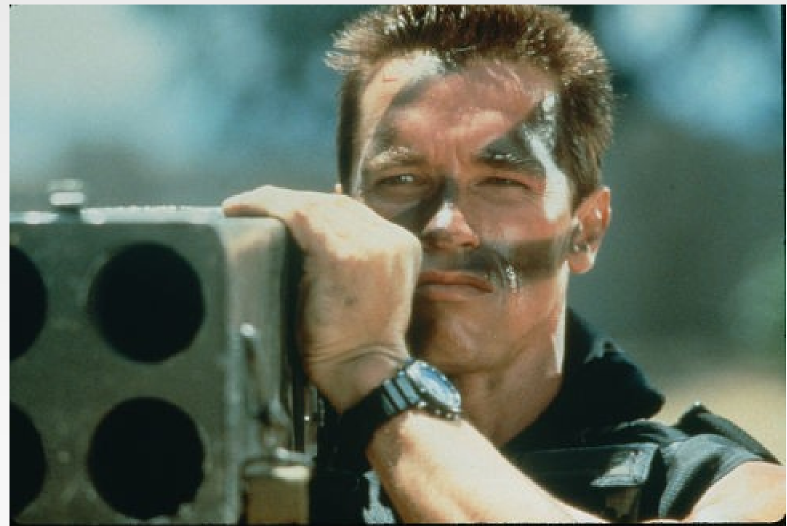
- DELETE FROM table → delete all rows in a table
- INSERT INTO table SELECT * FROM table2 → copy everything from table2 into table
  - Note that the columns must be in the same order and of the same data types!

Save and Undo

- The COMMIT command is used to actually write changes to the database
- ROLLBACK is used to undo changes that have no been committed
  - Changes are INSERT, UPDATE, and DELETE
  - CREATE TABLE cannot be rolled back
- Some DBs automatically commit after data definition commands, which means manipulation commands entered before them become committed also

The Third Piece of SQL

- Data creation → create table, insert
- Data manipulation → update, delete
- Now onto the actual “ask a question” (data query) part of SQL, the SELECT statement
- Feel the power!



SELECT

- The basic syntax →
  - SELECT columnlist
    FROM tablelist
    [WHERE conditions]
- * can be used for columnlist
- WHERE is optional but is where the power of the SELECT comes out
- If no rows meet the conditions of the WHERE nothing is returned

How it works

- Think of it as evaluating backwards
- First, figure out what rows meet the conditions in the table (or tables)
- Then, from those rows, display the columns specified

```
SELECT columnlist
FROM tablelist
[WHERE conditions]
```

Homework!

- Review: 1, 2, 3, 4, 5, 6, 7, 10, 11
  - For 10 and 11 just do these on EMP_1 since we didn't create an EMP_2 yet
- New project!

Comparisons & Operations

- =, >, >=, <, <=, <>
- IN
- AND, OR, NOT

Using (in)equalities

- With numbers they are as expected
  - SELECT * FROM students WHERE gpa > 3.5
- For strings they *strictly* compare left-to-right one character at a time
  - SELECT * FROM parts WHERE pnum > '5-XYZ'
- Dates are very tricky, best to use inequalities rather than equalities or DATEPART
  - SELECT * FROM orders WHERE orddate > '2009-03-25' AND orddate < '2010-03-25'
  - DATEPART(part, column)

IN

- If there is a set of values you want to compare against you use IN
  - SELECT * FROM customers WHERE state IN ('KY', 'OH')

Logical Operators

- AND and OR require expressions on each side and they will combine the answers from those expressions and give a boolean result
- AND → True only when both expressions are
- OR → False only when both expressions are
- NOT → reverse the true/false value of an expression

Practice

- Write a query to get all products with a price less than $100
- Additionally with no discount
- Write a query to get all products with a min not $5 and a qoh that is above 100 or less than 50 and only display the code and description
- All products that start their description with B&D

Answer 1

- Write a query to get all products with a price less than $100

- SELECT * FROM PRODUCT WHERE P_PRICE < 100

Answer 2

- Additionally with no discount

- SELECT * FROM PRODUCT WHERE P_PRICE < 100 AND P_DISCOUNT = 0

Answer 3

- Write a query to get all products with a min not $5 and a qoh that is above 100 or less than 50 and only display the code and description

- SELECT P_CODE, P_QOH FROM PRODUCT WHERE (P_QOH > 100 OR P_QOH) < 50 AND P_MIN <> 5

Arithmetic & Computed Columns

- Arithmetic can be used inside of queries
  - SELECT * FROM PRODUCT WHERE P_QOH > P_MIN*2
- Remember the discussion about whether to store 'computed columns'? Well we can created computed columns!
  - SELECT *, P_QOH-P_MIN FROM PRODUCT

Aliases

- Well that was kind of ugly, use an **alias** to get a prettier output
  - SELECT *, P_QOH-P_MIN AS surplus FROM PRODUCT

Practice

- Write a query that displays the product description, qoh, price, and "inventory value," which is defined as the qoh multiplied by the price
- Update the query so that it only returns inventory values over 1000

Answers

- SELECT P_DESCRIPT, P_QOH, P_PRICE, P_QOH*P_PRICE AS INV_VALUE FROM PRODUCT
- SELECT P_DESCRIPT, P_QOH, P_PRICE, P_QOH*P_PRICE AS INV_VALUE FROM PRODUCT WHERE INV_VALUE > 1000

Specialized Comparisons

- BETWEEN
- IS NULL
- LIKE
- IN
- EXISTS

BETWEEN

- Used to check if a value lies in a range
- Syntax → column BETWEEN(low, high)
  - age > 17 AND age < 36 becomes
    age BETWEEN (17, 36)
- Note that the low and high values are *noninclusive*

IS NULL

- NULLs are bad... why?
- IS NULL is the only operator that can be used appropriate with null values
- Syntax → column IS NULL
- Give me all the customers with no phone #:
  - SELECT * FROM customers WHERE phone IS NULL

LIKE

- Used for a 'fuzzy' comparison of strings
- Give me all the customers whose last name begin with K? → can't do this with equality
- Syntax → col LIKE 'some_string'
- Wildcards
  - % → match any number of characters, including none
  - _ → match a single character
- SELECT * FROM customers WHERE lastname LIKE 'K%'

Case Sensitivity

- SELECT * FROM customers WHERE lastname LIKE 'K%'
- What if someone typed their name all lower case? → 'kane'
- You can use UPPER and LOWER functions to convert strings to all upper or lower case
- SELECT * FROM customers WHERE UPPER(lastname) LIKE 'K%'

IN

- If there is a series of values you want to compare a column to this is what you use
- Syntax → column IN(list_of_values)
- SELECT * FROM customers WHERE LOWER(lastname) IN ('obama', 'bush', 'clinton')
- SELECT * FROM customers WHERE zip IN (41051, 41018, 41042, 41099)

EXISTS

- It's a bit odd, but EXISTS takes a query, runs it, and if it returns any rows executes the outside query
- Syntax → EXISTS(a_select_query)
- Give me the vendor information for all vendors but only if there are products in stock
  - SELECT * from VENDOR WHERE EXISTS(SELECT * FROM PRODUCT WHERE P_QOH > 0)

Practice

- Write a query to get all products with a min not $5 and a qoh that is above 100 or less than 50 and only display the code and description
- The products with no vendor code
- The above, along with having a price of less than $10
- Get all products that are hammers

Homework!

- Both the book and review questions listed below assume using EMP_2 and having the data as it is after question 11
- Review: 12, 15
- Get the last name and job code of all employees named John, June, or Alice
- Get all employees with no middle initial
- Get all employees hired after 1990, who don't have job code 500, and who have either an emp_pct less than 5 or an emp_num less than 105

Homework!

- All employees with an emp_pct greater than 3 but less than 9
- All employees who have been assigned a project number. Include a computed column named **foo** that is the emp_pct multiplied by the proj_num added to the emp_num
- The code to delete the table (you may not want to actually run this one!)