Week 2 Study Guide

Key Concepts:

|  |  |
| --- | --- |
| * RDB Concepts | * DML |
| * DDL * Set Ops * Aggregate functions * Stored procedures * Triggers * Transaction * ORM * Entity Framework * Normalization * INF, 2NF, 3NF | * Joins * Subqueries * Views * Functions * Isolation * ACID * ADO.NET (connected/disconnected) * db-first * repo.pattern * multiplicity * T-SQL |
|  |  |

**RDB concepts**

Database (DB) - A way to store data in a persistent way and be able to read and write it later

DBMS - Database Management System - A SQL server is an example, the database is actually stored in files.

RDB - Relational Database - Almost always some type of SQL - database structured to recognize relations among stored items of information.

RDBMS - An example is a SQL server

SQL is organized in rows (sometimes records) and columns. Together these are tables

The data in SQL is strong typed, so it must be declared what type.

Data in SQL also has constraints. This span beyond types and you can make almost any condition into a constraint

**Anomalies**

**Update anomaly** – If data items are scattered and are not linked to each other properly, then it could lead to strange situations. For example, when we try to update one data item having its copies scattered over several places, a few instances get updated properly while a few others are left with old values. Such instances leave the database in an inconsistent state.

**Deletion anomaly** – By deleting the record of one piece of data, we delete others records corresponding to that row. This may cause inconsistency between two related tables

**Insertion anomaly** - An Insert anomaly occurs when certain attributes cannot be inserted into the database without the presence of other attributes.

You can solve these anomalies with Normalization

**Normalization**

Normalization - Database normalization is the process of restructuring a relational database in accordance with a series of so-called normal forms in order to reduce data redundancy and improve data integrity.

Functional Dependency - dependency from one set of columns X to another set of columns Y eg. 1->1

If I know the values of the X columns, then there is exactly one possible set of Y values

\*Y is a fact about X

In our example, there is a dependency between Supplier and Phone #, and between Name and Color.

Syntax writing: Name ->(Color,Supplier,Phone#), Supplier ->(Phone#)

Candidate Key (concept in relation to FD)

Minimal set of columns that every other column depends on.

Can't remove any set without breaking the table

The values of any candidate key uniquely identify the row

In our example, Name is our candidate key. Other columns are not part of any Candidate key.

Composite Key

A name for a candidate key that is more than one column

Normal forms

3NF is normalized data

1NF requirements

1. atomic (indivisible) values. In our example Color should be broken up.

Try not to make more columns, make more rows

2. unique rows (using a primary key) - Primary Key is a chosen Candidate Key by the creator.

Sometimes you have to use a Composite key which is just a candidate key separated in multiple columns

In our example, we can make our Candidate key Name & Color to make sure every row is unique

2NF requirements

1. All columns that are not part of the candidate key must fully depend on every candidate key.

In our example, we must create a 2nd table for 2NF, splitting up the color column into a new table.

2. No partial dependencies on a candidate key

3NF requirements

1. No transitive dependencies on any candidate key. No non CK column can depend on another non CK column.

In our example, we should split our table a third time, to separate supplier and phone, because phone was dependent on supplier.

This creates a table of Supplier and Phone#. Only 1 CK in this third table because it is the minimal set of columns.

mnemonic for normal forms

non CKs "dependent on the keys, the whole keys, and nothing but the keys"

1NF - dependent on the keys

2NF - the whole keys

3NF - nothing but the keys

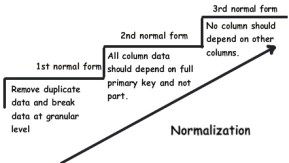
Once it is 3NF, all 3 anomalies will have been removed.

Normalized form can create tables that are harder to read.

Databases where it mostly is just reads and very low writes, denormalized tables can increase performance

**Pros and cons of a normalizing your database**

|  |  |
| --- | --- |
| Pros | Cons |
| **Updates run quickly** due to no data being duplicated in multiple locations. | **Since data is not duplicated, table joins are required**. This makes queries more complicated, and thus read times are slower. |
| **Inserts run quickly** since there is only a single insertion point for a piece of data and no duplication is required.  **Data integrity and consistency is an absolute must** if the database must be ACID compliant. A normalized database helps immensely with such an undertaking. | **Since joins are required, indexing does not work as efficiently**. Again, this makes read times slower because the joins don't typically work well with indexing. |



Foreign Key - establishes a relationship between 2 tables. It is only allowed to have values based on the primary key of the table it references.

Referential integrity (RI) is a relational database concept, which states that table relationships must always be consistent.

In other words, any foreign key field must agree with the primary key that is referenced by the foreign key.

Foreign Keys can point to Primary key in the same table

**T-SQL**

Microsoft uses Transact SQL. T-SQL. Data types will be slightly changed. Syntax has minor changes as well

Microsoft SQL Server is Microsoft implementation of a RDBMS, client side.

SQL server is any SQL server.

CRUD - Create Read Update Delete

SQL has a couple sub languages to it.

(LEARN THESE 8 statements below)

DML (Data Manipulation Language) - CRUD on row/records

INSERT - create statement

SELECT - read statement

UPDATE - update statement

DELETE and TRUNCATE- delete statement

DML DELETE FROM <table> (without `WHERE` clause) - goes through each row of the table and deletes it. table structure, constraints, etc. are not deleted.

DML TRUNCATE TABLE <table> - deletes all rows of the table all at once. faster than DELETE-without-WHERE, but does the same thing.

DDL (Data Definition Language) - CRUD on tables (cannot see individual rows)

CREATE - create statement

ALTER - update statement

DROP - delete statement

Drop vs Truncate – Truncate removes all rows from a table but leaves the table intact. Drop will completely remove the table from the database

DCL (Data Control Language) - deals with rights and permissions

GRANT- gives user’s access privileges to database.

REVOKE- withdraws user’s access privileges given by using the GRANT command.

DML

SELECT clauses - returns corresponding rows, if no answers, it will return empty string

SELECT <column names>, \* = all columns

FROM <table name> AS a (gives table name alias "a")

WHERE <column name> <condition>

GROUP BY - aggregates rows together. This allows us to select something else in the SELECT statement

GROUP BY <column names>

Aggregate functions take in many rows and return one value. (Homework)

HAVING happens after aggregation. (aka GROUP BY)

HAVING COUNT(State) >= 5

ORDER BY <column name>/<result> - sorts results. By default it orders by ascending order

ORDER BY COUNT(State) - normal order by

ORDER BY DESC COUNT(State) - descending order by

Execution flow of Select statement

FROM

ON

JOIN

WHERE

GROUP BY

WITH CUBE or WITH ROLLUP

HAVING

SELECT

DISTINCT

ORDER BY

TOP

INSERT Statement - allows us to add new rows to a table through inline data.

Can do more than 1 row at a time. Insert can also read from files

UPDATE and DELETE Statements - allows conditions. You can use the WHERE clause in conjunction.

DDL

CREATE Statement - creates a table. can change columns in the table as well

ALTER Statement - can add constraints to a table.

Constraints - how sql restricts anything in a db besides type. Enforced on columns

NOT NULL

(NULL)

PRIMARY KEY

FOREIGN KEY

CHECK <condition> - example check if a zip code string is 5 digits

UNIQUE - enforce a column is unique values

DEFAULT - if you don’t use default constraint, then the default value is null.

A column in SQL has a type, a name, and constraints

SQL Hierarchy

SQL Server > Database(s) > Schema (pretty much namespaces) > Tables

DDL edits at all levels Database and below

Alter table - allows you to add, delete or modify columns

Drop - deletes entire table

Truncate - deletes the rows but table still exists

Delete \* - will also delete every row from the table. row by row

Constraints

NOT NULL - Ensures that a column cannot have a NULL value

UNIQUE - Ensures that all values in a column are different

PRIMARY KEY - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table

FOREIGN KEY - Uniquely identifies a row/record in another table

CHECK - Ensures that all values in a column satisfies a specific condition

DEFAULT - Sets a default value for a column when no value is specified

INDEX - Used to create and retrieve data from the database very quickly by index

2 Types of INDEX- CLUSTERED AND NONCLUSTERED

A table can only have 1 clustered index. Defaults true on Primary Key

Can have many nonclustered indexes. Extra look up step over clustered. Defaults true on Foreign Key

Speeds up reads but slows down writes

Data Types

CHAR - fixed length non-unicode. works like a function: CHAR(10)

VARCHAR - dynamic length non-unicode. VARCHAR(10)

NCHAR - fixed length unicode. NCHAR(10). uses UNICODE UCS-2 character set

NVARCHAR - dynamic length unicode. NVARCHAR(10)

LIKE - Pattern match. Uses a form of regex. “a%” any string that begins with a, including a

WHERE CustomerName LIKE "regex"

BIT - 0 or 1. basically boolean type. 1 bit

TINY INT - 1 byte. In c# byte

SMALL INT - 2 byte. in C# short

INT - 4 bytes. In C# int

BIGINT - 8 bytes. in C# long

Operators

Conditionals have a lot of operators

AND, OR, NOT - basic

**IN - TRUE if the operand is equal to one of a list of expressions**

**EXISTS - TRUE if the subquery returns one or more records**

subquery example

SELECT id FROM Employees

WHERE HireDate > '2010-01-01' //this gives a list of ids hired after 2010

SELECT \* FROM Employees

WHERE ManagerID IN

(SELECT id FROM Employees

WHERE HireDate > '2010-01-01') //this gives a list of manager ids hired after 2010

single values. 1 value

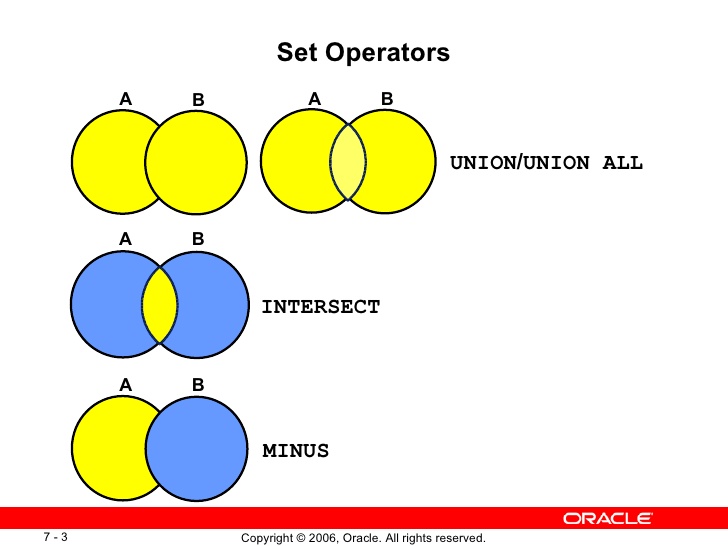
lists. N values

tables. N\*M values

The IN operator is used to evaluate a condition against a range of values

* EXISTS is much faster than IN, when the sub-query results is very large.
* IN is faster than EXISTS, when the sub-query results is very small.

**Set Operators**



Union - 2 or more sets combined subtract the duplicates

ex. SELECT <something>

UNION

SELECT <something else>

union all - 2 or more sets combined with duplicates. faster or equal to union

Intersect - All intersect between A & B

Intersect all - Intersect of 2 sets with all duplicates. 2 identical objects

Except - Non symmetrical

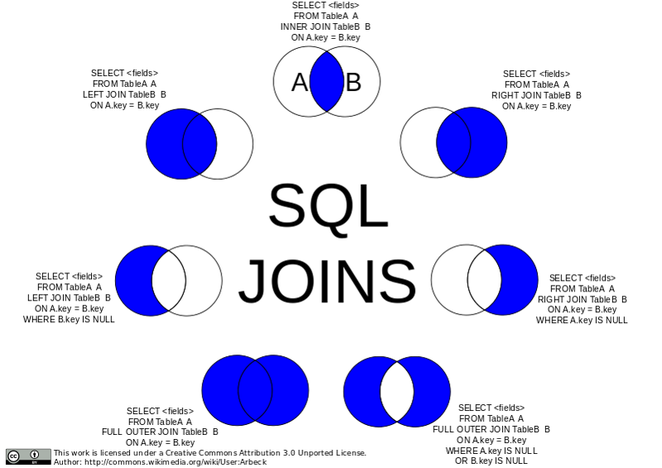
ex. a EXCEPT b

Except All - same as except

Use set operators if you had a managers table and employees table, and you could find out the names

of both sets.

Joins



SELF JOIN - A self JOIN is a regular join, but the table is joined with itself.

SELECT a.Name FROM Employees AS a INNER JOIN Employee AS b

ON a.ManagerID = b.ID

WHERE b.HireDate > '2010-01-01' //this also gives a list of manager ids hired after 2010

INNER JOIN - Work on 2 tables and selects records that have matching values in both tables

ex. INNER JOIN <table> ON <condition>

LEFT JOIN - Includes all records from the left table and also the matched records from the right table

if there is no match, the result is NULL on the right side

Always done on conditions, usually between PK and FK relations

FULL JOIN - returns all records when there is a match in either left or right. Right will fill with nulls when no match

CROSS JOIN - Each row from 1st table joins with all other rows of another table. There will be x\*y rows

Aggregate functions - take in a list and return one thing, needs group by (?)

AVG

COUNT

MIN

MAX

SUM

Other functions

YEAR(HireDate)

**Multiplicity**

one to one - (sales order header to sales order detail)

one to N - (crust to pizza) Pizza has a FK relationship to crust

N to N (pizza to topppings) - resolved with junction table (Pizza Topping Table with FKs to their respective tables)

possible challenge will invole joins and subqueries

OrderID Pizza Count

1 3

2 3

3 3

PizzaID OrderID

1 1

2 1

3 1

**SQL Injection (check connected files)**

When a program accepts input from the user and builds a SQL command without checking the input

Prevent injection by validating user inputs

Authentication can also stop SQL injection

In enterprise software, the database is usually the bottleneck of the execution because it can only support so many open connections

We want to do functional processing while the connection is closed

**Entity Framework**

It is a ORM. Object-Relational Mapper. This allows us to have a interface between object oriented data in C# and relational data in SQL

We can do operations on C# objects and have those get translated behind the scenes to SQl queries

EF has many workflows.

We will be using database-first approach to EF

Using a db we already made, EF will generate classes in c# for us

db-first approach vs code-first approach

learn the workflow. start in db and make their way to c# when you scaffold through entity framework

**ADO.NET (KNOW THESE FOR ADO.NET. \* are very important)**

ADO.NET separates data access from data manipulation into discrete components that can be used separately or in tandem. ADO.NET includes .NET Framework data providers for connecting to a database, executing commands, and retrieving results.

Connected arch.

Connection object - key object

Command object- key object

DataReader\* - key object

Execute reader - key method

Execute nonquery - key method

DataReader

Provides connected forward-only, read-only access to the data source. It is optimized for speed. The DataReader is instantiated through a Command object.

Disconnected arch. - key object

Connection - key object

(Command) - key object

DataAdapter\* - key object (using data reader but we dont see it)

DataSet\* - key object

Fill - key method

Update - key method

DataSet

Provides a consistent way to deal with disconnected data completely independently of the data source. The DataSet is essentially an in-memory relational database, serving as a container for the DataTable, DataColumn, DataRow, Constraint, and DataRelation objects.

The XML format serializes and transports a DataSet. A DataSetcan be accessed and manipulated either as XML or through the methods and properties of the DataSet interchangeably; theXmlDataDocument class represents and synchronizes the relational data within a DataSet object with the XML Document Object Model (DOM).

Data Adapter

Bridges the data source and the disconnected DataSet or DataTable classes. The DataAdapter wraps the connected classes to provide this functionality. It provides a method to retrieve data into a disconnected object and a method to reconcile modified data in the disconnected object with the data source.

Entity Framework - ORM

DBContext\* - key object (our own derived classes)

DBSet\* - key object (contains one table worth of stuff)

Our own model classes - key object (created by scaffolding)

SaveChanges - key method

Add - key method

Update - key method

DbContext generally represents a database connection and a set of tables. DbSet is used to represent a table.

SQL

Computed Column - we can make a column have a computed value based on other columns

SELECT First, Last FROM Person

WHERE (Middle IS NULL) OR (Middle = '');

Go to Week2FridayWarmup.sql

**ACID**

A Transaction is a single unit of work.

Every statement in SQL is a transaction

We can also make explicit transactions from multiple statements

If there is an error in the middle of a statement, it will roll back

Each statement will either entirely succeeds or entirely fails

Transactions should follow some certain theories

ACID Properties

A - Atomicity.

All or nothing, no partial effects. Indivisible

C - Consistency.

Don't violate the constraints of the database.

I - Isolation

Transactions don't interfere with each other.

From the point of the view of the programmed transactions, it has exclusive use of the system.

Expensive for SQL Server to promise this property

|  |  |
| --- | --- |
| Isolation Levels | anomaly/problem fixed |
| read\_uncommitted | (minimum, doesn't implement isolation) |
| read\_committed | dirty read (sees unfinished changes of other transactions) |
| Repeatable | non-repeatable read (read same row twice within a transaction and sees 2 different values because of other transaction) |
| Serializable | phantom reads (read a table twice and see extra rows the 2nd time) |

As you move up Isolation levels, you have more overhead, but more isolation (less problems)

This is implemented by placing locks are certain parts of tables.

This creates a possibility of deadlock.

D - Durability

When the transaction is done, all changes to the system are permanent.

Persistent, not in volatile memory.

**repo.pattern**

http://blog.gauffin.org/2013/01/repository-pattern-done-right/

repo will be a class that manages dbcontext for us.

It will expose simple straight forward methods. (gets, sets, edit, delete) implements CRUD

It will talk to our library

It will use the scaffolded classes to communicate to the database

we seperate the classes we have made to the database access

provides abstraction from the details of the database

Console is connected to Data Access just to create DBContext

**Views**

In SQL, a view is a virtual table based on the result-set of an SQL statement.

A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

You can add SQL functions, WHERE, and JOIN statements to a view and present the data as if the data were coming from one single table.

A view always shows up-to-date data! The database engine recreates the data, using the view's SQL statement, every time a user queries a view.

**Stored Procedures vs. Functions**

|  |  |
| --- | --- |
| Stored Procedures (SP) | User Defined Functions (UDF) |
| SP can return zero, single or multiple values | Function must return a single result (can be a one data point or a table) |
| We can use transaction in SP | We can’t use transaction in UDF |
| SP can have input and output parameters | UDF can have only input parameters |
| We can call FUNCTION from SP | Can’t call SP from function |
| We can’t use SP in SELECT/WHERE/HAVING statements | You can use a UDF in SELECT/WHERE/HAVING statements |
| We can use exception handling using Try-Catch block in SP | We can’t use Try-Catch blocks in UDF |