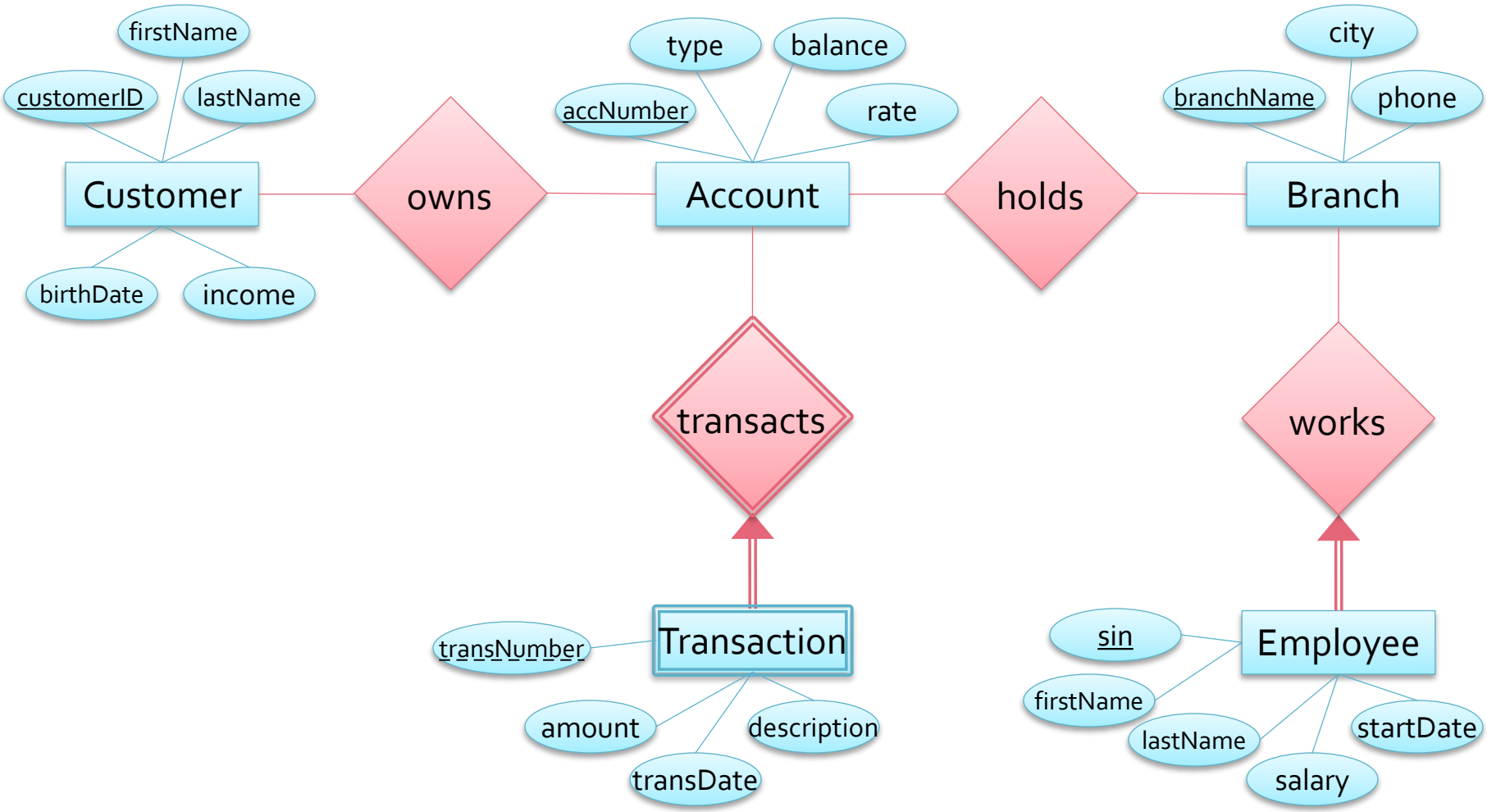
CMPT 354

SQL Constraints and Triggers

SQL Constraints

- Constraints
  - Primary Key
  - Foreign Key
  - General table constraints
  - Domain constraints
  - Assertions
- Triggers

Bank ERD


firstName
customerID
lastName
Customer
birthDate
income
owns
type
balance
accNumber
rate
Account
transacts
transNumber
Transaction
amount
description
transDate
holds
city
branchName
phone
Branch
works
sin
Employee
firstName
lastName
salary
startDate

Bank Schemata

- Customer = {*customerID*, *firstName*, *lastName*, *birthDate*, *income*}
- Account = {*accNumber*, *type*, *balance*, *rate*, *branchName*}
  - *branchName* is a foreign key referencing Branch
- Owns = {*customerID*, *accNumber*}
  - *customerID* and *accNumber* are foreign keys referencing Customer and Account
- Transaction = {*accNumber*, *transNumber*, *amount*, *transDate*, *description*}
  - *accNumber* is a foreign key referencing Account
- Branch = {*branchName*, *city*, *phone*}
- Employee = {*sin*, *firstName*, *lastName*, *salary*, *startDate*, *branchName*}
  - *branchName* is a foreign key referencing Branch

Primary and Foreign Keys

Primary Key Constraints

- Every table should have a primary key
- When a primary key constraint is created it specifies that:
  - The attributes of the primary key cannot be null
  - The primary key must be unique
- Violating a primary key causes the violating update to be rejected

SQL Server Primary Keys

- A primary key constraint can be specified in a number of ways
  - Create or alter table statement
    - PRIMARY KEY (<attribute>)
  - Selecting the attribute(s) and choosing the primary key menu item in the context menu
- By default SQL server creates a clustered index on the primary key attributes
  - Clustered indexes support range queries

Foreign Key Constraints

- Represents a relationship between two tables
- If table *R* contains a *foreign key*, on attributes {*a*}, that references table *S*
  - {*a*} must correspond to the primary key of *S*
    - Must have the same number of attributes, and
    - The same domains
  - Values for {*a*} in *R* must also exist in *S* except that
    - If {*a*} is not part of the primary key of *R* it may be *null*
  - There may be values for {*a*} in *S* that are not in *R*

Foreign Key Specification

- Foreign keys specify the actions to be taken if referenced records are updated or deleted
  - For example, create a foreign key in Account that references Branch
    - Assign accounts of a deleted branch to Granville
    - Cascade any change in branch names

```
…
branchName CHAR(20) DEFAULT 'Granville',
FOREIGN KEY (branchName) REFERENCES Branch
        ON DELETE SET DEFAULT
        ON UPDATE CASCADE)
```

Cascading Changes

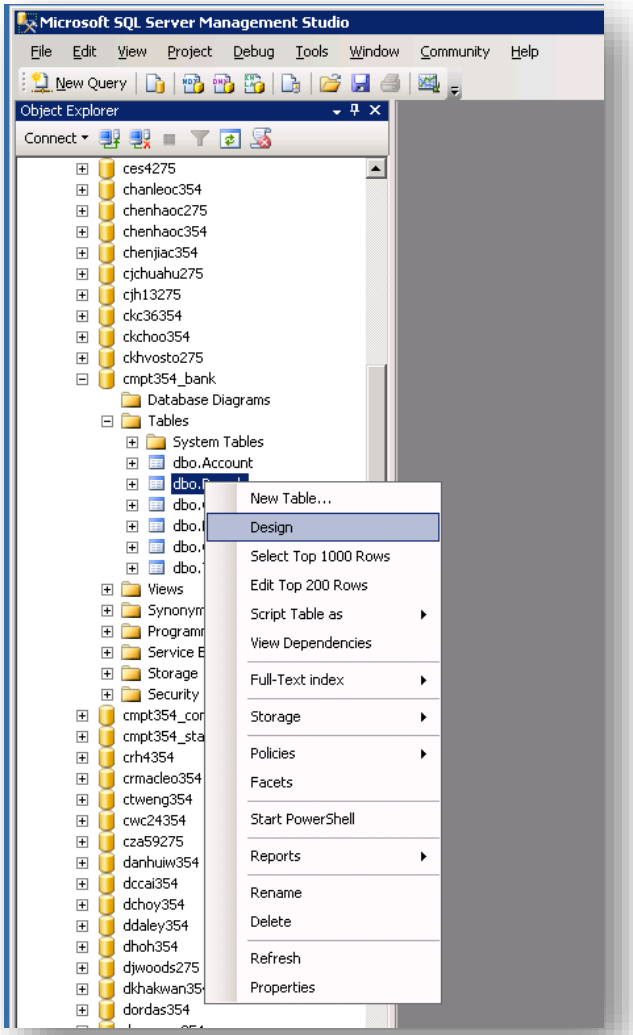
- It is possible that there can be a chain of foreign key dependencies
  - e.g. branches, accounts, transactions
- A cascading deletion in one table may cause similar deletions in a table that references it
  - If any cascading deletion or update causes a violation, *the entire transaction* is aborted

Foreign Keys in SQL Server

- Foreign keys can be specified in a number of different ways
  - A separate foreign key statement
    - Very similar to the SQL standard
  - A *References* statement following a column name
  - Using the GUI
- Updates and deletions in the referenced table can be set to a number of actions
  - No action (the default) – the transaction is rejected
  - Cascade, set null or set default

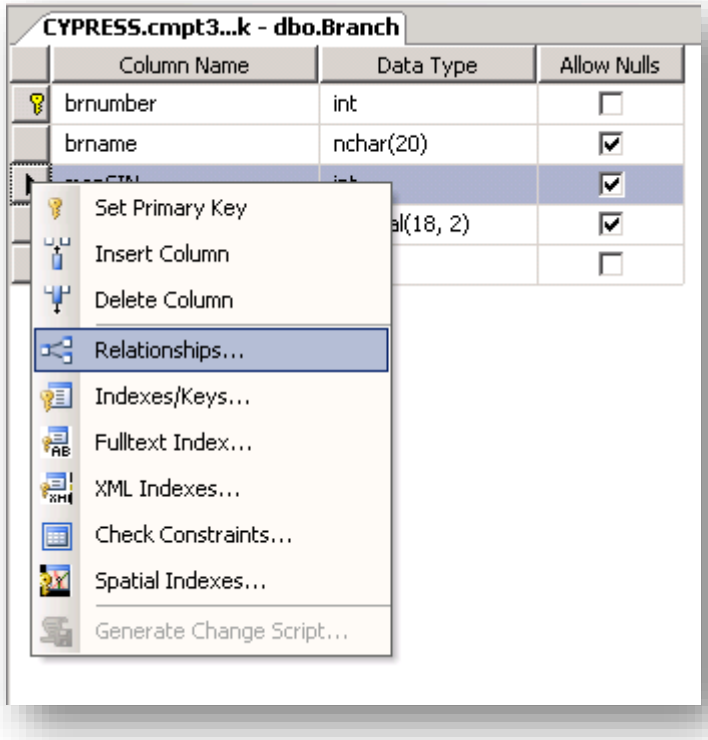
Constraints in SQL Server

- To use the GUI to create constraints in SQL Server
  - Select Design on the desired table using the context menu
- This opens design view for the selected table
  - The schema can be changed
  - Constraints can be added

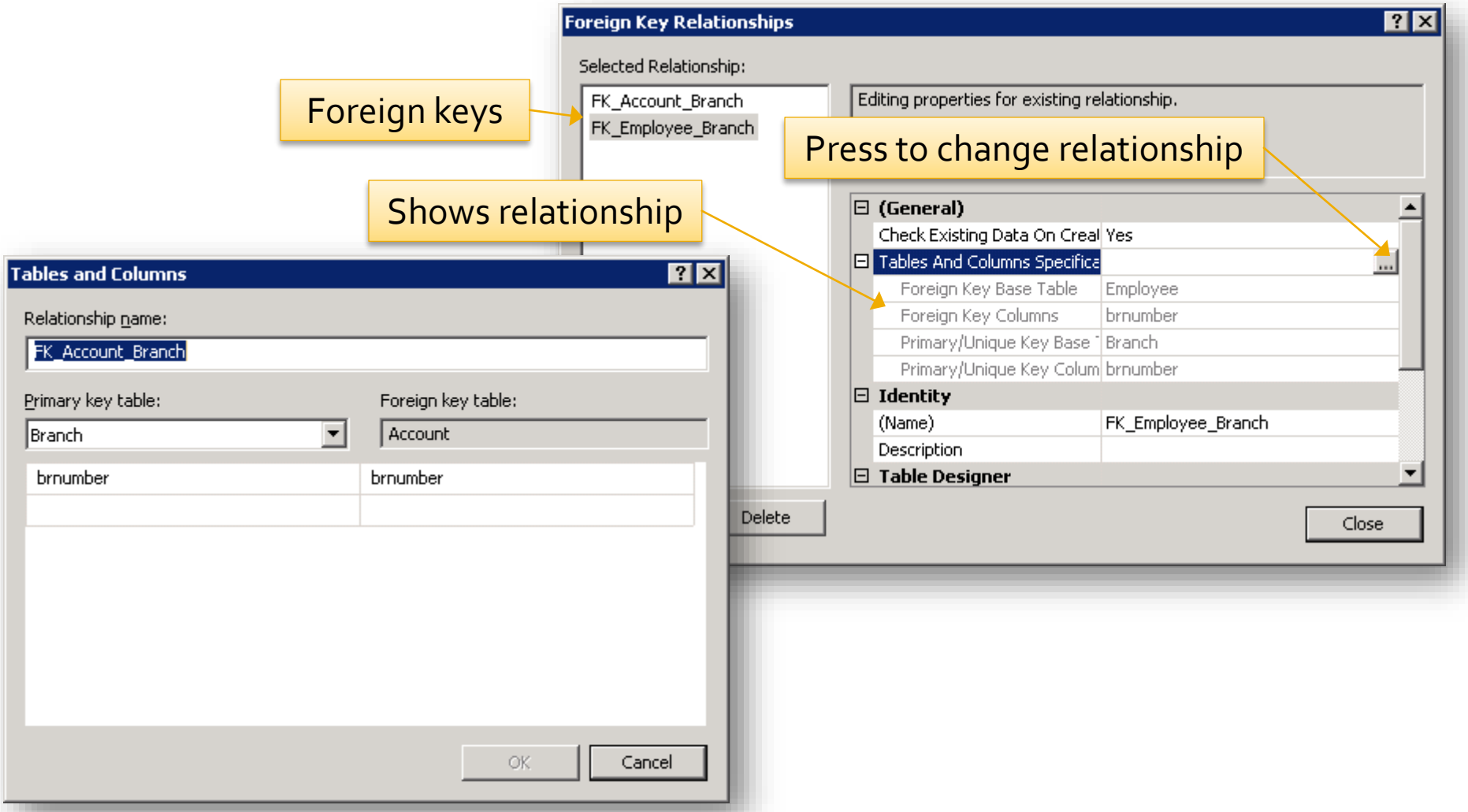


Constraints in SQL Server

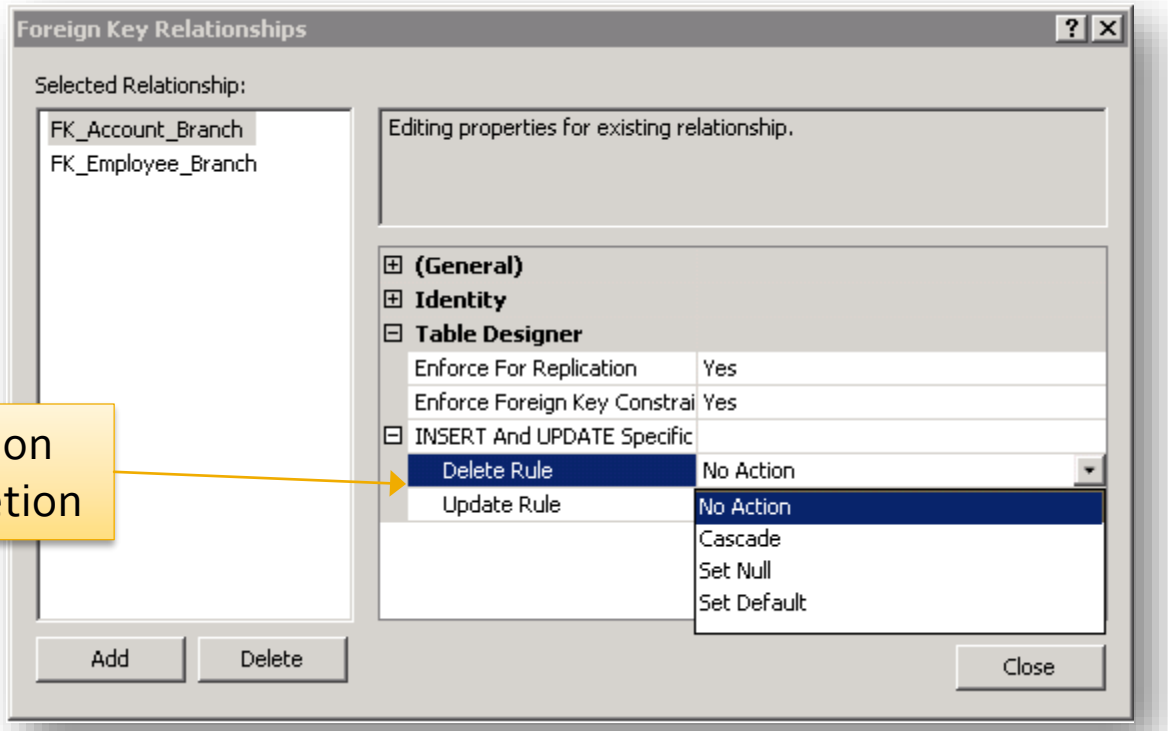
- To add a constraint
  - Use the context menu
  - And select the desired constraint type
- Primary key constraints are added with no more input
- For other constraints, more input is required



Relationships in SQL Server



Relationships in SQL Server



Specify action on update and deletion

Referencing non Primary Keys

- By default SQL foreign keys reference the primary key (of the referenced table)
- It is possible to reference a list of attributes
  - The list must be specified after the name of the referenced table
  - The specified list of attributes must be declared as a candidate key of the referenced table (UNIQUE)
    - In SQL Server in Create or Alter Table statement or
    - Select Indexes/Keys from the context menu

General Constraints

General Constraints

- A general or *table* constraint is a constraint over a single table
  - Included in a table's **CREATE TABLE** statement
  - Table constraints may refer to other tables
- Defined with the **CHECK** keyword followed by a description of the constraint
  - The constraint description is a Boolean expression, evaluating to true or false
  - If the condition evaluates to false the update is rejected

Constraint Example

- Check that a customer's income is greater than zero, and that customer ID is not equal to an employee SIN

```
CREATE TABLE Customer
    (customerID          CHAR(11),
     ...,
    income               REAL,
     PRIMARY KEY (customerID),
     CONSTRAINT zeroIncome CHECK (income > 0),
     CONSTRAINT notEmp CHECK (customerID NOT IN
          (SELECT sin FROM Employee)))
```

this wouldn't work in SQL Server

SQL Server Check Constraints

- SQL Server check constraint are limited to comparisons with *scalar expressions*
  - That is, expressions that contain single values
  - Expressions that contain SQL queries are therefore not allowed
- Check expressions can include comparisons to results of User Defined Functions (UDFs)
  - As long as the function returns a scalar value

UDF

- A UDF is an SQL Server function
- UDFs purpose and output vary widely
  - They can return scalar values or tables
  - They can be written in T-SQL or a .NET language
- UDF are useful for a number of reasons
  - Modular programming
  - Faster execution
  - Reduced network traffic

UDFs and Check Constraints

```
CREATE FUNCTION checkEmpNotCustomer()
RETURNS int
AS
BEGIN
        DECLARE @result int
        if (select COUNT(*) from Employee e, Customer c
            where c.customerID = e.sin) = 0
                SET @result = 1
        ELSE
                SET @result = 0
        RETURN @result
END
```

The corresponding check expression is dbo.checkEmpNotCustomer = 1

Warning: Check constraints that call UDFs introduce overhead

Note that this constraint should also be added to the Customer table

Domains and Types

Domain Constraints

- New domains can be created using the **CREATE DOMAIN** statement
  - Each such domain must have an underlying source type (i.e. an SQL base type)
  - A domain must have a name, base type, a restriction, and a default optional value
    - The restriction is defined with a **CHECK** statement
- Domains are part of the DB schema but are not attached to individual table schemata

Domain Constraint Example

- Create a domain for minors, who have ages between 0 and 18
  - Make the default age 10 (!)

```
CREATE DOMAIN minorAge INTEGER DEFAULT 10
  CHECK (VALUE > 0 AND VALUE <= 18)
```

Using Domain Constraints

- A domain can be used instead of one of the base types in a CREATE TABLE statement
  - Comparisons between two domains are made in terms of the underlying base types
    - e.g. comparing an age with an account number domain simply compares two integers
- SQL also allows distinct types to be created
  - Types are distinct so that values of different types cannot be compared

Creating Types

- The SQL **CREATE TYPE** clause defines new types
  - To create distinct ID and account number types:
    - **CREATE TYPE ID AS INTEGER**
    - **CREATE TYPE Accounts AS INTEGER**
  - Assignments, or comparisons between ages and account numbers would now be illegal
    - Although it is possible to *cast* one type to another

SQL Server Base Types

bigint	binary(n)	bit	char(n)
8 bytes	fixed length binary	single bit	fixed length
date	datetime	datetime2	datetimeoffset
yyyy-mm-dd	date and time	larger range	includes time zones
decimal	float	image	int
select precision, scale	approximate	variable length binary	4 bytes
money	nchar(n)	ntext	numeric
8 byte monetary	fixed length unicode	variable length unicode	equivalent to decimal
nvarchar(n | max)	real	smalldatetime	smallint
variable length unicode	4 byte float	limited range	2 bytes
smallmoney	sql_variant	text	time
4 byte monetary	allows many types	variable length	time (not date)
tinyint	uniqueidentifier	varbinary(n | max)	varchar(n | max)
1 byte	used for keys	variable length binary	variable length

Creating SQL Server Types

- SQL Server allows types to be created using syntax similar to the SQL standard
  - CREATE TYPE SSN FROM varchar(11);

Assertions

Assertions

- Table constraints apply to only one table
- Assertions are constraints that are separate from CREATE TABLE statements
  - Similar to domain constraints, they are separate statements in the DB schema
  - Assertions are tested *whenever the DB is updated*
    - Therefore they may introduce significant overhead
- Transact-SQL (MS SQL) does not implement assertions

Example Assertion

- Check that a branch's assets are greater than the total account balances held in the branch

```
CREATE ASSERTION assetCoverage
CHECK (NOT EXISTS
        (SELECT *
         FROM Branch B
         WHERE assets <
                (SELECT SUM (A.balance)
                 FROM Account A
                 WHERE A.branchName = B.branchName)))
```

This prevents changes to both the Branch and Account tables

Assertion Limitations

- There are some constraints that cannot be modeled with table constraints or assertions
  - What if there were participation constraints between customers and accounts?
    - Every customer must have at least one account and every account must be held by at least one customer
  - An assertion *could* be created to check this situation
    - But would prevent new customers or accounts being added!

Triggers

Triggers

- A trigger is a procedure that is invoked by the DBMS as a response to a specified change
  - A DB that has a set of associated triggers is sometimes referred to as an *active database*
  - Triggers are available in most current commercial DB products
    - And are part of the SQL standard
- Triggers carry out *actions* when their triggering conditions are met
  - Generally SQL constraints only *reject* transactions

Why Use Triggers?

- Triggers can implement business rules
  - e.g. creating a new loan when a customer's account is overdrawn
- Triggers may also be used to maintain data in related database tables
  - e.g. Updating derived attributes when underlying data is changed, or maintaining summary data
- Many trigger actions can also be performed by *stored procedures*

Trigger Components

- Event
  - A specified modification to the DB
    - May be an insert, deletion, or change
    - May be limited to specific tables
    - A trigger may *fire* before or after a transaction
- Condition
- Action

Trigger Components

- Event
- Condition
  - A Boolean expression or a query
    - If the query answer set is non-empty it evaluates to true, otherwise false
    - If the condition is true the trigger action occurs
- Action

Trigger Components

- Event
- Condition
- Action
  - A trigger's action can be very far-ranging, e.g.
    - Execute queries
    - Make modifications to the DB
    - Create new tables
    - Call host-language procedures

Trigger Syntax

- The SQL standard gives a syntax for triggers
  - In practice, trigger syntax varies from system to system
  - Many features of triggers are common to the major DBMS products, and the SQL standard

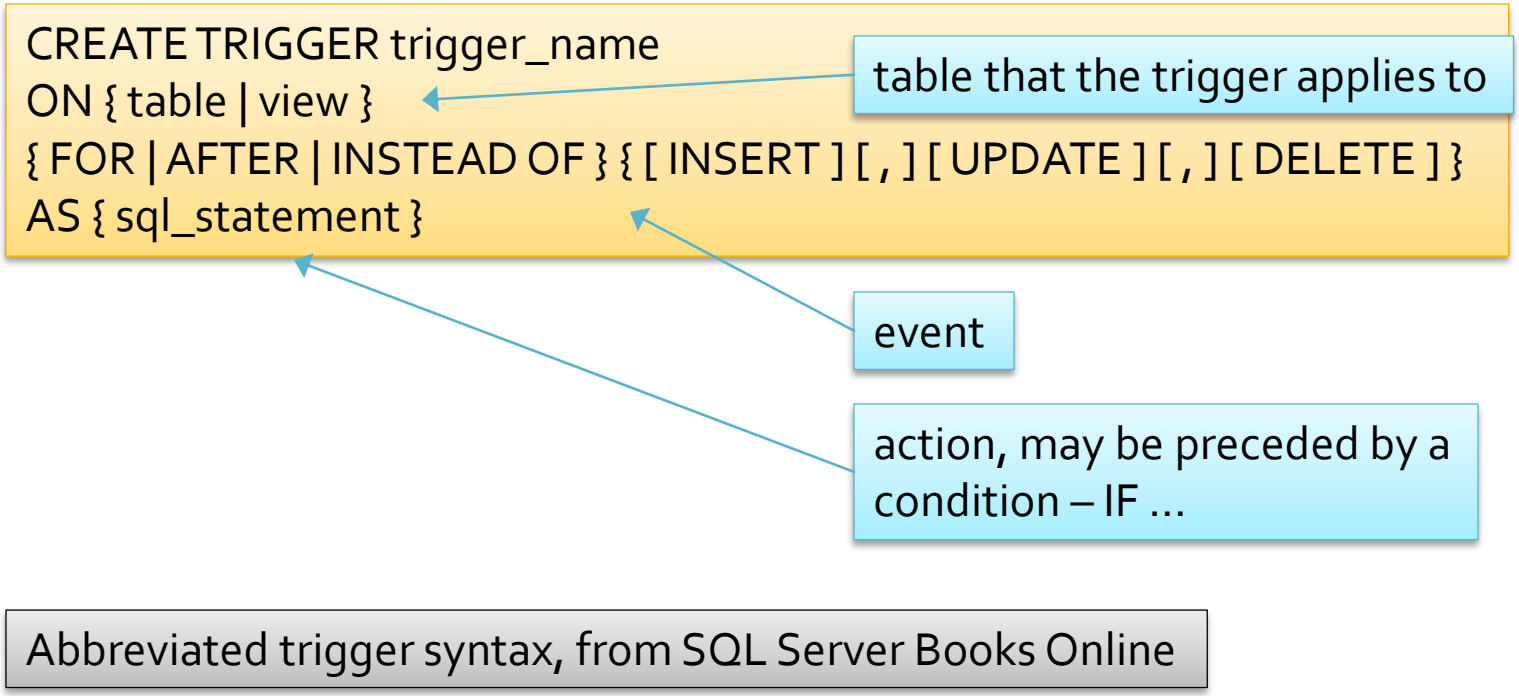
SQL Server Trigger Example

- Write a trigger to send a message when customer data is added or changed
- The trigger has three components
  - Event – insert or update of the Customer table
  - Condition – always!
  - Acton – print a message
    - By printing an SQL Server error message

```
CREATE TRIGGER reminder1
ON Sales.Customer
AFTER INSERT, UPDATE
AS RAISERROR ('Notify Customer Relations', 16, 10);
```

Example from SQL Server Books Online

SQL Server Trigger Syntax



Abbreviated trigger syntax, from SQL Server Books Online

SQL Server Trigger Events

- Triggers can apply to any change to a table
  - Deletion, insertion or update
  - Or any combination of the three
- Triggers can be specified as
  - AFTER
    - The default (sometimes referred to as *for* triggers)
    - Occur after the transaction has taken place
  - INSTEAD OF
    - The trigger *replaces* the triggering statement
    - Only one instead of trigger is allowed for each insert, update or delete statement for a table

After Trigger Order

- After triggers only fire after referential cascade actions and constraint checks
  - Such constraints checks must succeed
    - Otherwise the transaction would not take place
- Specifically the order is
  - Enforce constraints
  - Enforce foreign key constraints
  - Create *inserted* and *deleted* tables
  - Execute the triggering statement
  - Execute *after* trigger

Trigger Actions

- A trigger's action can be almost anything
  - Here is a second version of the reminder trigger that sends an email

```
CREATE TRIGGER reminder2
ON Sales.Customer
AFTER INSERT, UPDATE, DELETE
AS
EXEC msdb.dbo.sp_send_dbmail
        @profile_name = 'AdventureWorks Administrator',
        @recipients = 'danw@Adventure-Works.com',
        @body = 'Don''t forget to print a report for the sales force.',
        @subject = 'Reminder';
```

Example from SQL Server Books Online

Trigger Conditions

- Trigger conditions are contained in an if statement
  - IF should be followed by a Boolean expression
  - The Boolean expression is commonly an SQL expression
    - e.g. IF EXISTS(subquery)
- If statements in triggers may have an else clause

The Boss is Richer

- Reject the insertion of new employees whose salary is greater than their manager

```
CREATE TRIGGER TR_Richer_Boss
ON Employee AFTER INSERT
AS
IF EXISTS
(SELECT *
 FROM Manager m, Employee e
 WHERE e.brNumber = m.brNumber AND e.salary > m.salary)
BEGIN
    ROLLBACK TRANSACTION
    RAISERROR('Error: salary too high', 16 ,1)
END
```

Could also be implemented as a check constraint with a UDF

inserted and deleted

- When a table is changed temporary tables are created containing the changed data
  - The *inserted* table contains records that were inserted by a transaction
  - The *deleted* table contains records that were removed by a transaction
  - If a table is updated the *inserted* table contains the new rows and the *deleted* table the old rows

Balancing Balance

- Add the amount of a transaction to the balance of the associated account

```
CREATE TRIGGER TR_Account_Balance
ON Transactions AFTER INSERT
AS
BEGIN
    UPDATE account
    SET balance = balance + (SELECT amount FROM inserted)
END
```

This only works if there is only one transaction being inserted

Balancing Balance Again

- Add the amount of a transaction to the balance of its account – corrected version

```
ALTER TRIGGER TR_Account_Balance
ON Transactions AFTER INSERT
AS
BEGIN
    UPDATE account SET balance = ins.newBalance FROM
    (SELECT a.accnumber,
        a.balance + SUM(i.amount) AS newBalance
     FROM Account a INNER JOIN inserted i
       ON i.accNumber = a.accNumber
     GROUP BY a.accNumber, a.balance) AS ins
     WHERE account.accnumber = ins.accnumber
END
```

Trigger Summary

- Triggers can be used for many purposes
  - Enforcing business rows
  - Adding functionality to the database
  - …
- Triggers do carry overhead
  - Constraints should be enforced by PKs, FKs and simple check constraints where possible
  - Triggers should be made as efficient as possible

Indexes

Introduction to Indexes

- An index is a data structure that provides efficient access to records
  - Based on the value of one or more attributes
  - An index on a database is conceptually similar to an index at the back of a book
    - The index provides data to efficiently locate a record in a table
    - Without reading the entire table

Table Scans

- Database tables are typically resident on secondary storage such as hard drives
  - A table may be too large to fit in main memory
- The basic method for finding a record it to read the table from disk
  - Referred to as scanning a table
- This is a very inefficient way of finding one or two records in a large table

Indexes

- An index is a secondary data structure that maps attribute values to record IDs
  - A record ID contains the disk address of a record
    - Indexes typically allow records to be looked up with a handful of disk reads
- Typical OLTP indexes are variations of two data structures
  - Hash tables
  - (Binary) search trees

Hash Indexes

- A hash index is a hash table where the hash function maps attribute values to record IDs
  - Hash indexes are very fast
    - Typically requiring one or two disk reads
- A good hash function is uniform and random
  - So cannot map a range of attribute values to related locations in the hash table
  - Therefore hash indexes cannot be used for range searches

Tree Indexes

- A tree index directs searches in much the same way as a binary search tree
  - In practice, databases do not use *binary* search trees
  - The common implementation is a B tree (or variant) which is an *n*-ary tree structure
    - Where the value of n is derived from the size of attribute values, record IDs and disk pages
- Tree indexes are not as fast as hash indexes
  - But do support range searches

Clustered Indexes

- A database table may be supported by a number of indexes
  - One, and only one, of these indexes can be a clustered index
- A clustered index is one where the underlying data file is sorted on the index search key
  - That is, the attribute used to look up index entries
  - Clustered indexes provide much more support for range queries than un-clustered indexes
    - Provided that the index is a tree index

There's no such thing as a free lunch

- Indexes increase the efficiency of operations requiring look-up of attribute values
- There is a cost to maintaining indexes
  - Every insertion and deletion requires modification of each index on the table
  - Updates may require modification of indexes

Which Attribute?

- Database tables can support multiple indexes
  - With single attribute search keys, or
  - Compound (multi-attribute) search keys
- The DB administrator is responsible for determining which indexes to create
  - By analyzing table work loads
  - Modern DBMS have tools to aid in determining which indexes are appropriate for a table
    - Index tuning

SQL Server 2008 allows 1 clustered and 999 un-clustered indexes per table