School Management System

Group 1

Group Members:
SHIH-YUAN WANG, XIAOZHU ZHANG

Instructor:
Arthur Paruzel

CONTENTS

Problem Description

PART ONE

To design a database to maintain information about:

Enrollment System

Departments
Instructors
Students
Courses
Classrooms

Sections

conflict of time and classroom / instructor and time / department

Enrollment

course validation / capacity / repeated enrollment / time conflict

Dynamic enrollment process

capacities & spots left

Financial System

Tuition

Audit table of tuition

Tuition status

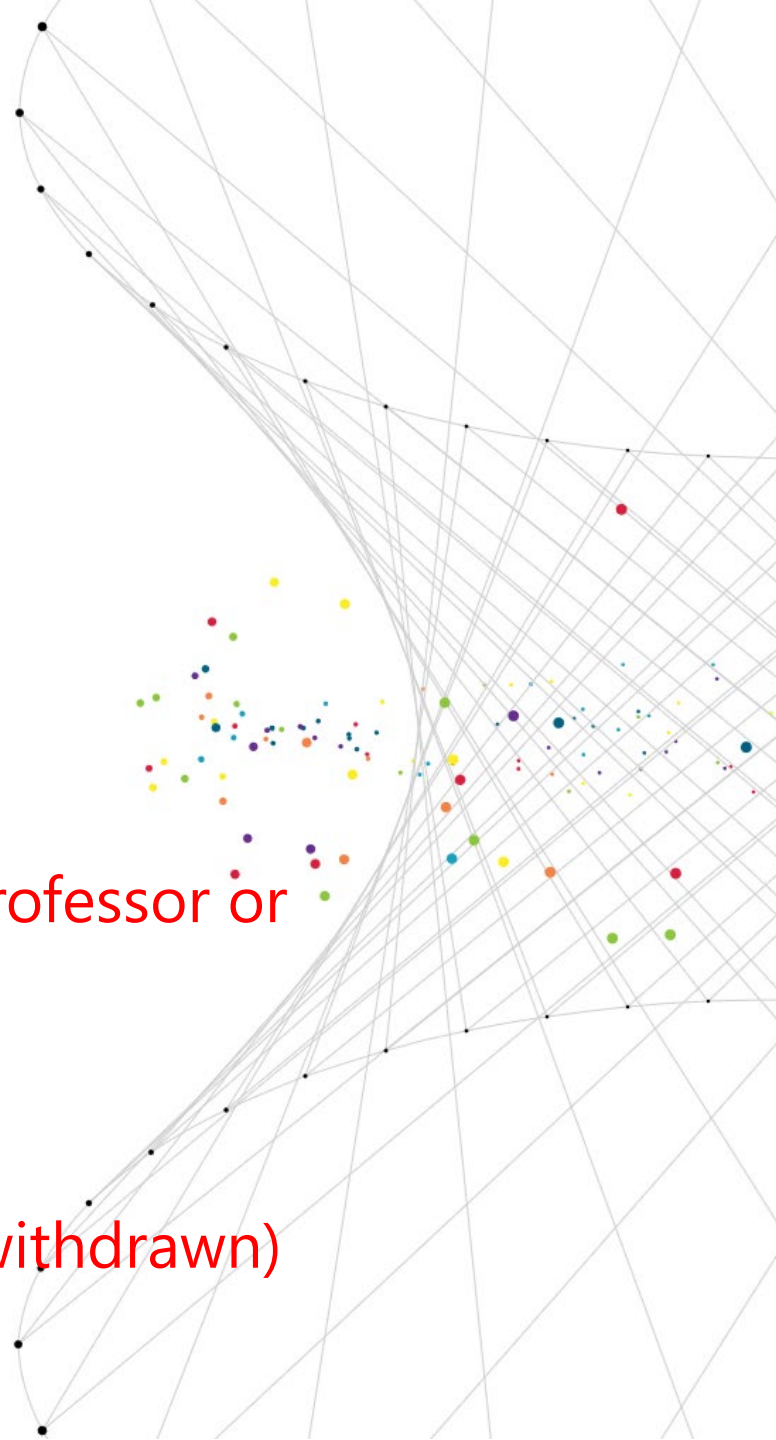
deficit, equilibrium, or surplus.
courses the student takes → tuition payable
deficit → warned and dropped

Payment

salary (+) = courses fees + titles (Professor, Associate Professor or Assistant Professor) + years of service
withdraw (-)

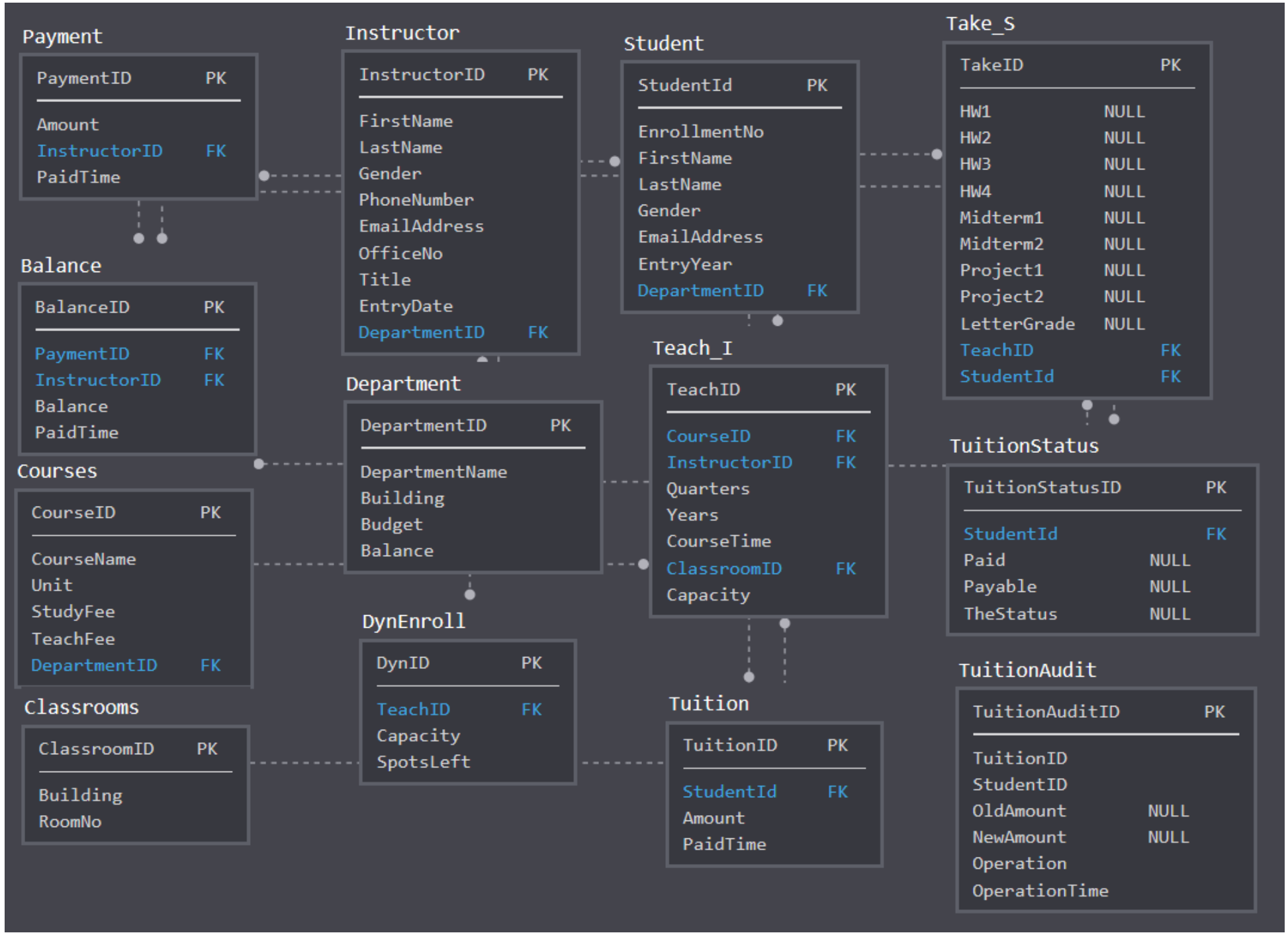
Balance

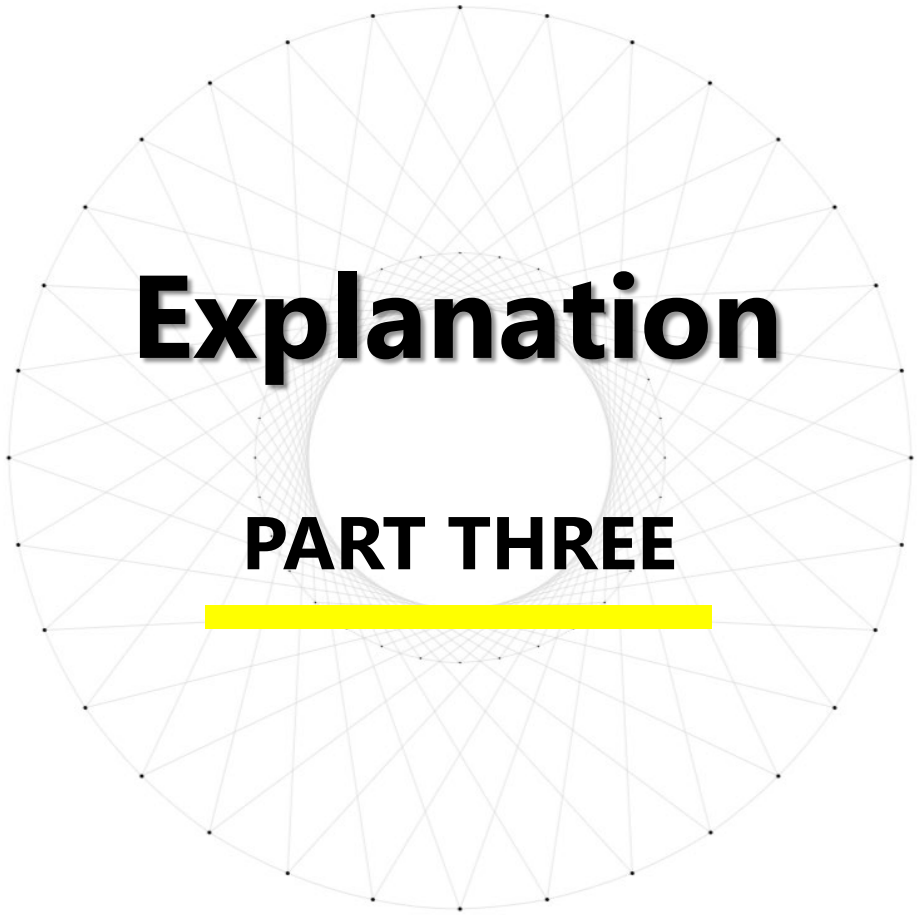
Balance = (previous balance) + (salary paid) – (money withdrawn)



Solution Proposal

PART TWO


Payment
PaymentID PK
Amount
InstructorID FK
PaidTime
Instructor
InstructorID PK
FirstName
LastName
Gender
PhoneNumber
EmailAddress
OfficeNo
Title
EntryDate
DepartmentID FK
Student
StudentId PK
EnrollmentNo
FirstName
LastName
Gender
EmailAddress
EntryYear
DepartmentID FK
Take_S
TakeID PK
HW1 NULL
HW2 NULL
HW3 NULL
HW4 NULL
Midterm1 NULL
Midterm2 NULL
Project1 NULL
Project2 NULL
LetterGrade NULL
TeachID FK
StudentId FK
Balance
BalanceID PK
PaymentID FK
InstructorID FK
Balance
PaidTime
Department
DepartmentID PK
DepartmentName
Building
Budget
Balance
Teach_I
TeachID PK
CourseID FK
InstructorID FK
Quarters
Years
CourseTime
ClassroomID FK
Capacity
TuitionStatus
TuitionStatusID PK
StudentId FK
Paid NULL
Payable NULL
TheStatus NULL
Courses
CourseID PK
CourseName
Unit
StudyFee
TeachFee
DepartmentID FK
DynEnroll
DynID PK
TeachID FK
Capacity
SpotsLeft
Classrooms
ClassroomID PK
Building
RoomNo
Tuition
TuitionID PK
StudentId FK
Amount
PaidTime
TuitionAudit
TuitionAuditID PK
TuitionID
StudentID
OldAmount NULL
NewAmount NULL
Operation
OperationTime



Explanation

PART THREE

Enrollment System - Teaching Check

TeachCheck: instead of insert trigger
conflict of time and classroom / instructor and time / department

```
CREATE TRIGGER TeachCheck
ON Teach_I
INSTEAD OF INSERT
AS
BEGIN
    -- conflict of time and classroom
    declare @num1 int
    set @num1 = (select count(*)...
    if(@num1 > 0)
    select getdate() as ErrorTime, 'Classroom is occupied at the time!' as Error

    -- conflict of instructor and time
    declare @num2 int
    set @num2 = (select count(*)...
    if(@num2 > 0)
    select getdate() as ErrorTime, 'Instructor cannot teach 2 classes at the same time!' as Error

    -- instructor cannot teach the class in department he/she is not in
    declare @a int
    declare @b int
    set @a = (select c.DepartmentID...
    set @b = (select ins.DepartmentID...
    if (@a != @b)
    select getdate() as ErrorTime, 'Instructor cannot teach the class in department he/she is not in!' as Error

    if(@num1 = 0 and @num2 = 0 and @a = @b)...
END
```

Enrollment System - Student Enrollment 1

ListOfClasses2019: view
Show the list of available classes in 2019

```
CREATE VIEW ListOfClasses2019
AS
select t.TeachID as TeachID,
       c.CourseName as Course, c.Unit as Units, d.DepartmentName as Department,
       i.FirstName + ' ' + i.LastName as Instructor, cr.Building + ' ' + cr.RoomNo as Locations,
       t.Quarters + ' ' + t.Years as Term, t.CourseTime as CourseTime,
       t.Capacity, dy.SpotsLeft
from Teach_I t join Course c on t.CourseID = c.CourseID
     join Department d on c.DepartmentID = d.DepartmentID
     join Instructor i on t.InstructorID = i.InstructorID
     join Classroom cr on t.ClassroomID = cr.ClassroomID
     join DynEnroll dy on t.TeachID = dy.TeachID
where t.Years = '2019'
```

Enrollment System - Student Enrollment 2

ChangeSpotsLeft: after insert trigger
Update the number of spots left after a student take the class

```
CREATE TRIGGER ChangeSpotsLeft
ON Take_S
AFTER INSERT
AS
BEGIN
    declare @dyn int
    set @dyn = (select SpotsLeft from DynEnroll
    where TeachID = (select TeachID from inserted))
    set @dyn = @dyn - 1

    update DynEnroll
    set SpotsLeft = @dyn
    where TeachID = (select TeachID from inserted)
END
```

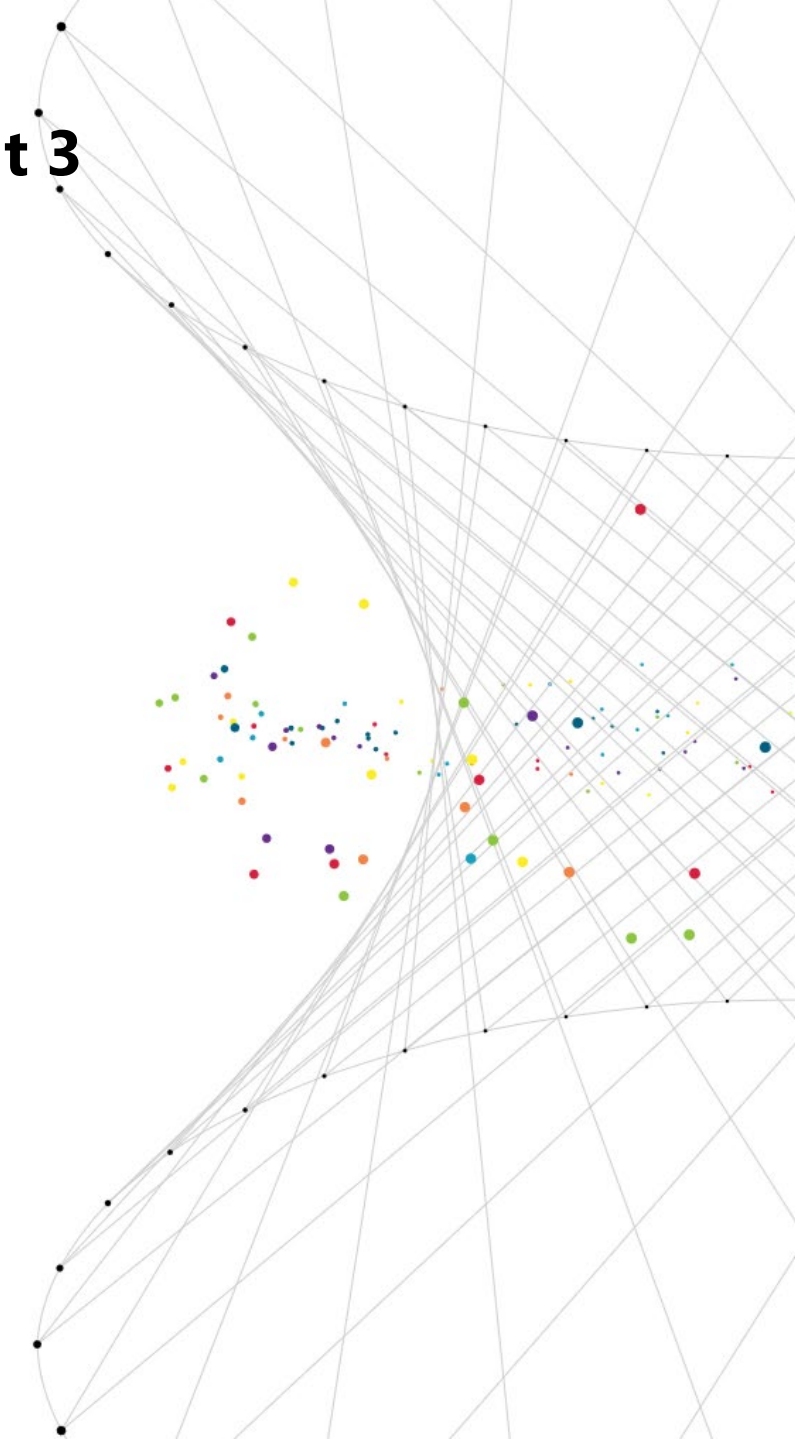
ChangeSpotsLeft1: after delete trigger
Update the number of spots left after a student drop the class

```
CREATE TRIGGER ChangeSpotsLeft1
ON Take_S
AFTER DELETE
AS
BEGIN
    declare @dyn int
    set @dyn = (select SpotsLeft from DynEnroll
    where TeachID = (select TeachID from deleted))
    set @dyn = @dyn + 1

    update DynEnroll
    set SpotsLeft = @dyn
    where TeachID = (select TeachID from deleted)
END
```

Enrollment System - Student Enrollment 3

EnrollCheck: instead of insert trigger
course validation / capacity / repeated enrollment / time conflict

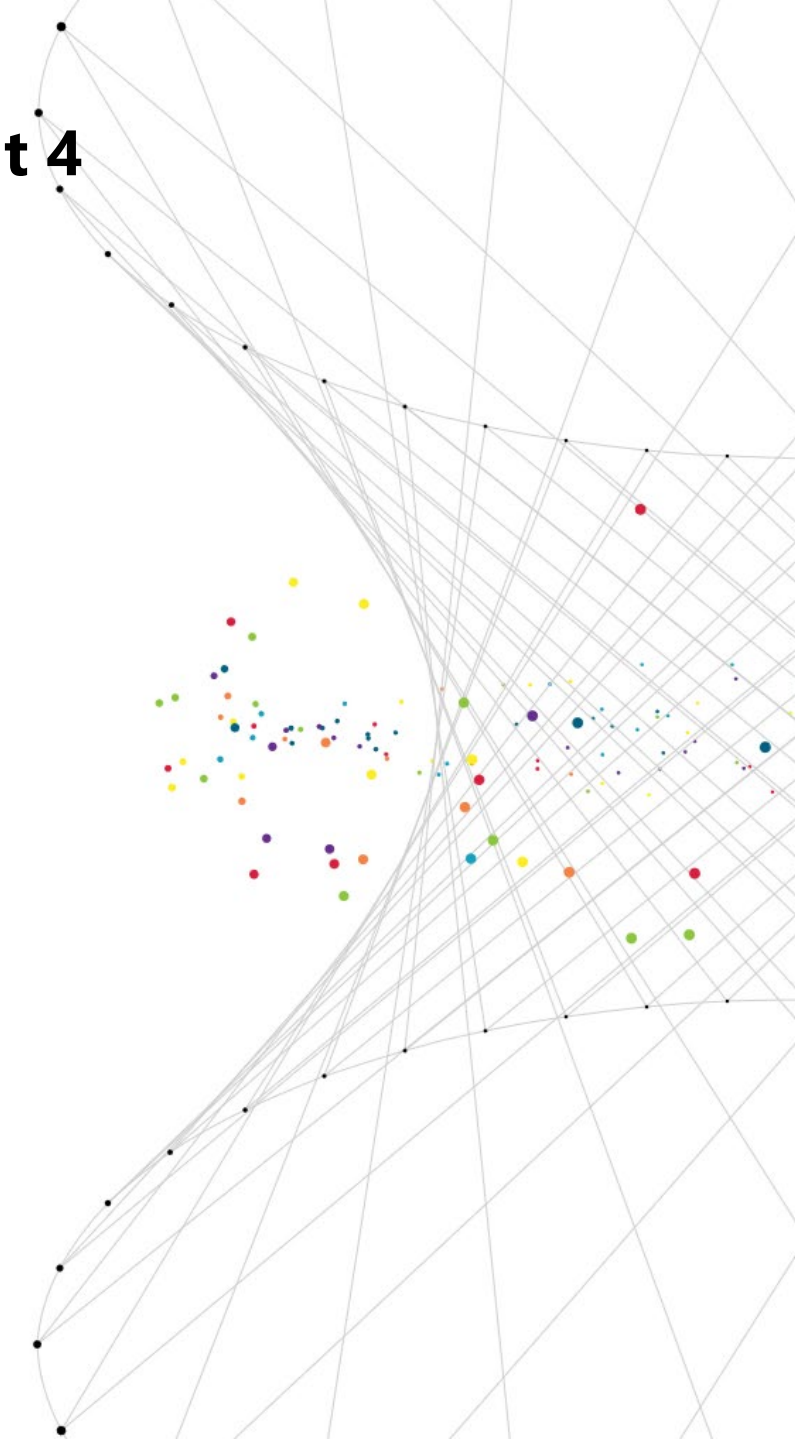
```
CREATE TRIGGER EnrollCheck
ON Take_S
INSTEAD OF INSERT
AS
BEGIN
    -- course validation check
    declare @num3 int
    set @num3 = (select count(*)...
    if(@num3 = 0)
    select GETDATE() as ErrorTime, 'No such courses in 2019!' as Error
    -- capacity check
    declare @dyn int
    set @dyn = (select SpotsLeft from DynEnroll where TeachID = (select TeachID from inserted))
    if (@dyn = 0)
    select GETDATE() as ErrorTime, 'The class is Full!' as Error
    -- repeated enrollment check
    declare @num2 int
    set @num2 = (select count(*)...
    if(@num2 != 0)
    select GETDATE() as ErrorTime, 'Already enrolled!' as Error
    -- time conflict check
    declare @num1 int
    set @num1 = (select count(*)...
    if(@num1 != 0)
    select GETDATE() as ErrorTime, 'Time Conlict!' as Error

    if(@dyn != 0 and @num1 = 0 and @num2 = 0 and @num3 != 0)...
END
```



Enrollment System - Student Enrollment 4

Enrollment: stored procedure
Input ①Student ID, ②Section ID (TeachID), ③'take' or 'drop'

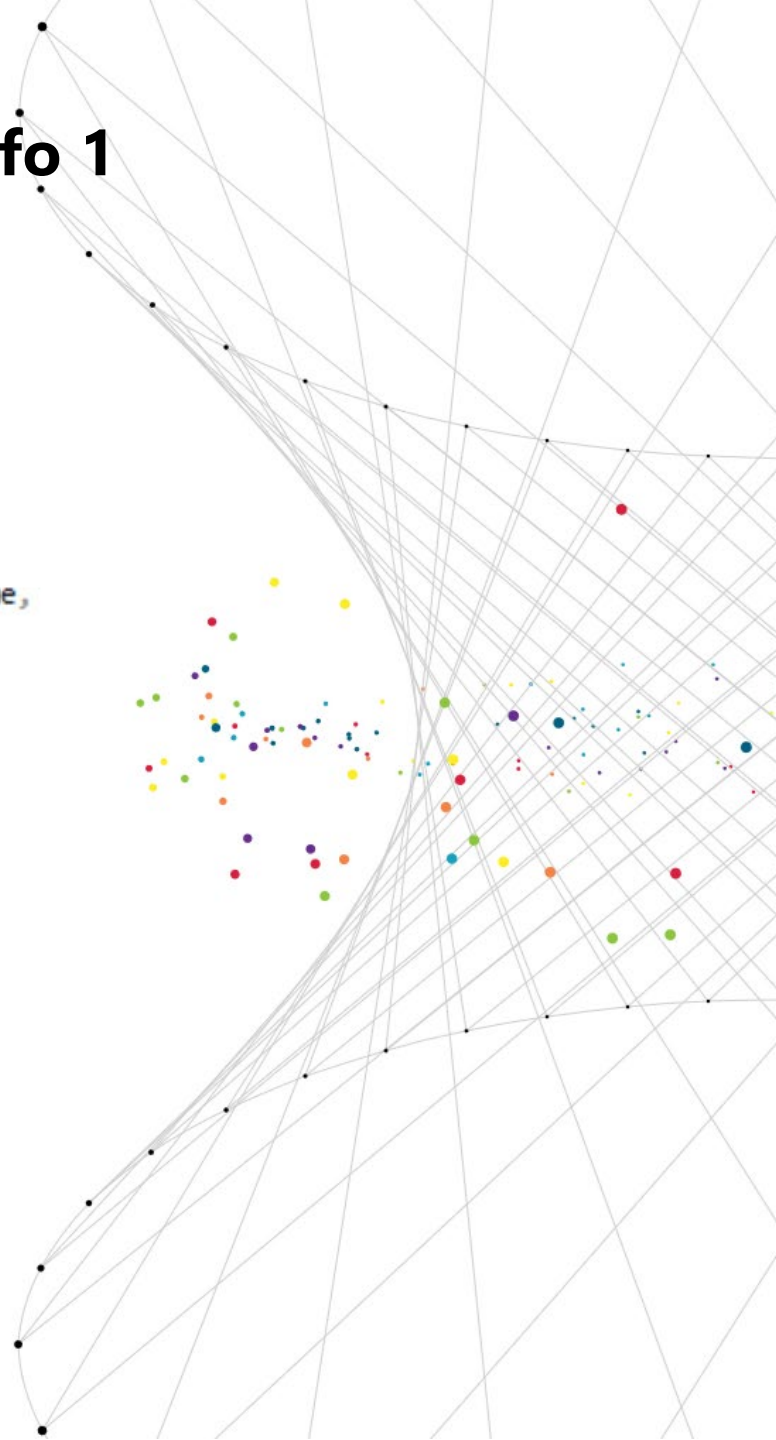
```
CREATE PROCEDURE Enrollment @StudentID int, @TeachID int, @select nvarchar(10)
AS
BEGIN

if(@select = 'take')
insert into Take_S(StudentID, TeachID, LetterGrade) values(@StudentID, @TeachID, 'NA')

if(@select = 'drop')
delete from Take_S
where StudentID = @StudentID and TeachID = @TeachID

-- Show class information after enrollment or drop
select StudentID, ts.TeachID as TeachID, Course, Units, Department,
       Instructor, Locations, Term, CourseTime
from Take_S ts join ListOfClasses2019 lc on ts.TeachID = lc.TeachID
where StudentID = @StudentID

END
```



Enrollment System - Instructor Courses Info 1

InstructorCoursesInfo: stored procedure
Input ①Instructor ID, ②Years, ③Quarters

```
create procedure InstructorCoursesInfo @InstructorID int, @Years char(4), @Quarters varchar(6)
AS
BEGIN
    select ti.InstructorID, i.FirstName +' '+ i.LastName as InstructorName, ti.CourseID, c.CourseName,
       c.Unit, ti.CourseTime, d.DepartmentName, cl.Building+ ' ' + cl.RoomNo as Locations
    from Teach_I as ti
    inner join Course as c on c.CourseID = ti.CourseID
    inner join Instructor as i on ti.InstructorID = i.InstructorID
    inner join Department as d on c.DepartmentID = d.DepartmentID
    inner join Classroom as cl on cl.ClassroomID = ti.ClassroomID
    where ti.InstructorID = @InstructorID and ti.Years =  @Years and ti.Quarters = @Quarters
END
```



Enrollment System - Instructor Courses Info 2

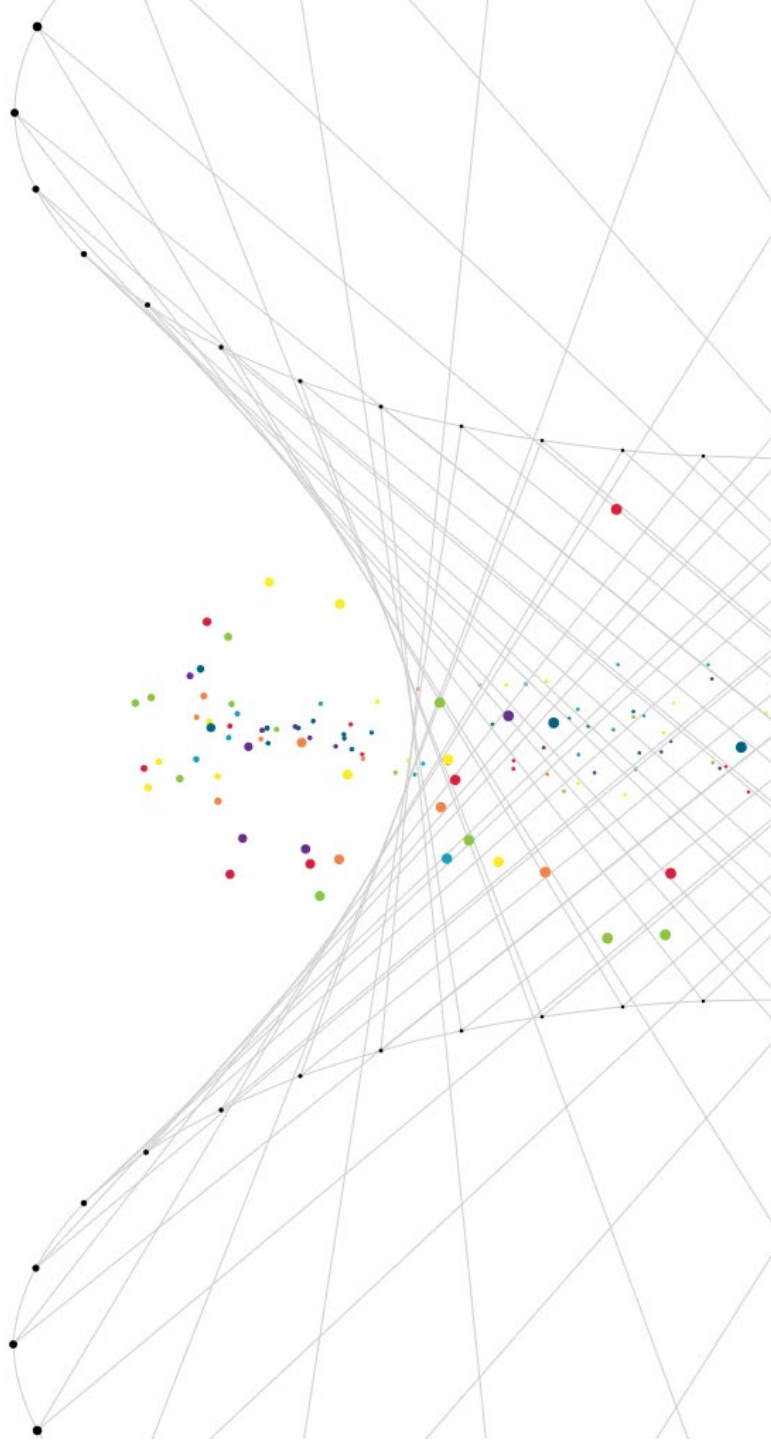
Rollbook: stored procedure
Input Section ID (TeachID)

```
CREATE PROCEDURE RollBook @TeachID int
AS
BEGIN
-- Show enrolled students' grades
select ts.TakeID as TakeID, s.StudentID as StudentID, s.FirstName +' '+ s.LastName as StudentName,
       ti.TeachID as TeachID, c.CourseName as CourseName,
       ts.HW1 as HW1, ts.HW2 as HW2, ts.HW3 as HW3, ts.HW4 as HW4, ts.Midterm1 as Midterm1,
       ts.Midterm2 as Midterm2, ts.Project1 as Project1, ts.Project2 as Project2,
       ts.Final as Final, ts.LetterGrade as LetterGrade
from Take_S ts join Teach_I ti on ts.TeachID = ti.TeachID
     join Student s on ts.StudentID = s.StudentID
     join Course c on ti.CourseID = c.CourseID
where ts.TeachID = @TeachID

-- Show the number of enrolled students
Select count(TakeID) as [Number of Enrolled Students]
from Teach_I as ti
join Take_S as ts on ti.TeachID = ts.TeachID
where ts.TeachID = @TeachID
```

Enrollment System - Update Grade 1

ShowUpdatedGrade: after update trigger
Show the updated grades

```
CREATE TRIGGER ShowUpadatedGrade
ON Take_S
AFTER UPDATE
AS
BEGIN
SET NOCOUNT ON;

select i.TakeID as TakeID, s.StudentID as StudentID, s.FirstName +' '+ s.LastName as StudentName,
       ti.TeachID as TeachID, c.CourseName as CourseName,
       ts.HW1 as HW1, ts.HW2 as HW2, ts.HW3 as HW3, ts.HW4 as HW4, ts.Midterm1 as Midterm1,
       ts.Midterm2 as Midterm2, ts.Project1 as Project1, ts.Project2 as Project2,
       ts.Final as Final, ts.LetterGrade as LetterGrade, 'Updated' as [Status]
from inserted i join Take_S ts on i.TakeID = ts.TakeID
     join Teach_I ti on ts.TeachID = ti.TeachID
     join Student s on ts.StudentID = s.StudentID
     join Course c on ti.CourseID = c.CourseID

END
```



Enrollment System - Update Grade 2

UpdateGrade_XXX: stored procedure
Input ①TakeID, ②Grade

```
-- HW1
CREATE PROCEDURE UpdateGrade_HW1 @TakeID int, @HW1 numeric(4,2)
AS
BEGIN
update Take_S set HW1 = @HW1 where TakeID = @TakeID
END
```

Notice: there are 10 stored procedures to update grades
for HW1, HW2, HW3, HW4, Midterm1, Midterm2, Project1, Project2, Final, Letter Grade,
respectively

Financial System – Tuition 1

TuitionAudit_insert:
after insert trigger

```
CREATE TRIGGER TuitionAudit_insert
    ON dbo.Tuition
    AFTER INSERT
AS
BEGIN
    insert into TuitionAudit
    select i.TuitionID, i.StudentID,
    null, i.Amount, 'insert', GETDATE()
    from inserted i join Tuition t
    on i.TuitionID = t.TuitionID
END
```

TuitionAudit_delete:
after delete trigger

```
CREATE TRIGGER TuitionAudit_delete
    ON dbo.Tuition
    AFTER DELETE
AS
BEGIN
    insert into TuitionAudit
    select d.TuitionID, d.StudentID,
    d.Amount, null, 'delete', GETDATE()
    from deleted d
END
```

TuitionAudit_update:
after update trigger

```
CREATE TRIGGER TuitionAudit_update
    ON dbo.Tuition
    AFTER UPDATE
AS
BEGIN
    insert into TuitionAudit
    select i.TuitionID, i.StudentID,
    d.Amount, i.Amount, 'update', GETDATE()
    from inserted i join Tuition t
    on i.TuitionID = t.TuitionID join deleted d
    on d.TuitionID = t.TuitionID
END
```

Financial System – Tuition 2

TuitionCheck1: stored procedure
Check the tuition status of one student

```
CREATE PROCEDURE TuitionCheck1 @StudentID int
AS
BEGIN
    declare @Paid numeric(10,2) -- Total paid tuition
    set @Paid = (select sum(Amount) from Tuition
    where StudentID = @StudentID)

    declare @Payable numeric(10,2)  -- Total tuition(StudyFee) payable
    declare @StagingVar numeric(10,2)
    set @StagingVar = (select sum(c.StudyFee)...
    set @Payable = isnull(@StagingVar,0)

    IF EXISTS (select StudentID from TuitionStatus...
END
```

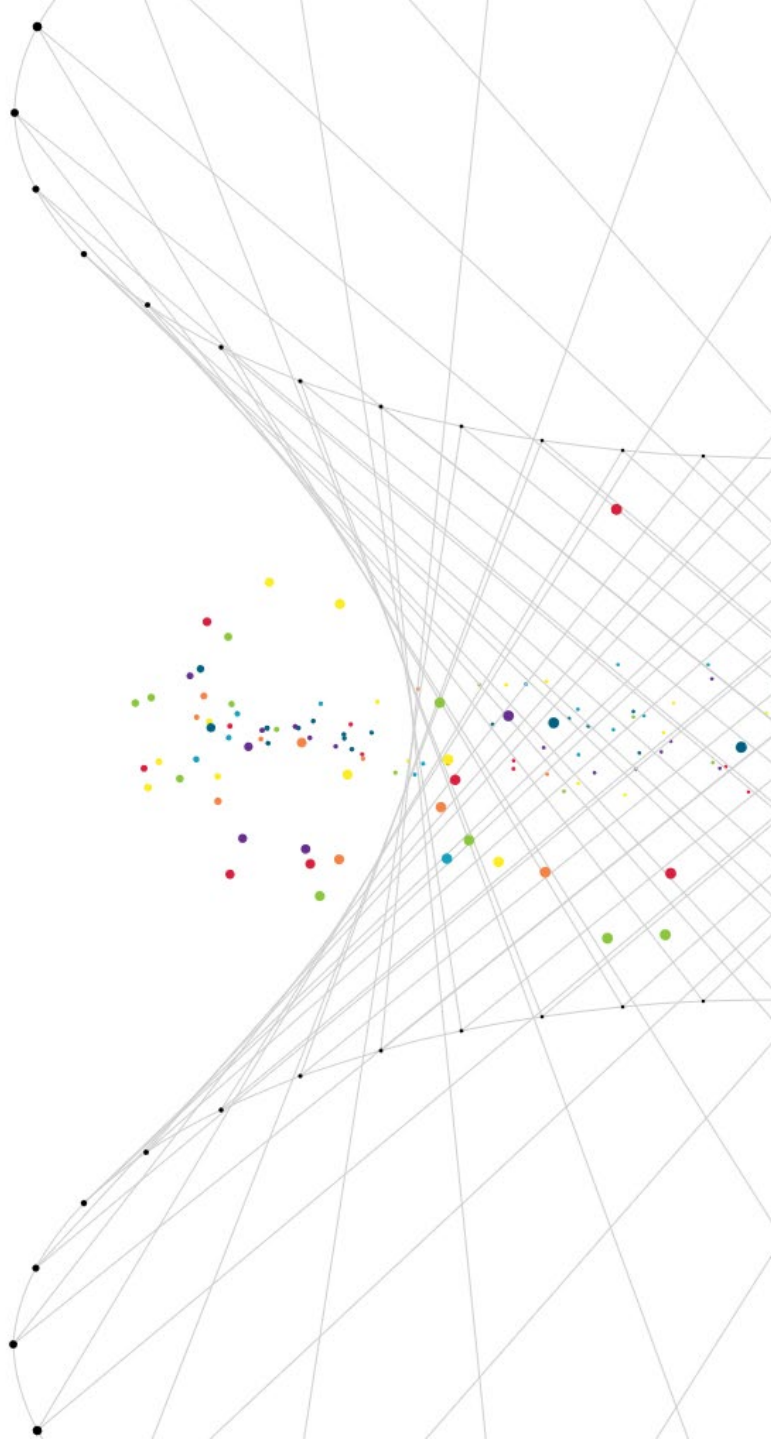
TuitionCheckAll: stored procedure
Check the tuition status of all students

```
CREATE PROCEDURE TuitionCheckAll
AS
BEGIN
    declare @max int
    set @max = (select max(StudentID) from Student)

    declare @i int
    set @i = (select min(StudentID) from Student)
    WHILE (@i <= @max)
    BEGIN
    IF EXISTS (select StudentID from Student where StudentID = @i)
       BEGIN
           exec TuitionCheck1 @i       -- insert or update tuition status
           set @i = @i + 1
       END
    END
END
```

Financial System – Tuition 3

DropStudentsinDeficit: stored procudure
Drop students who did not pay enough tuition

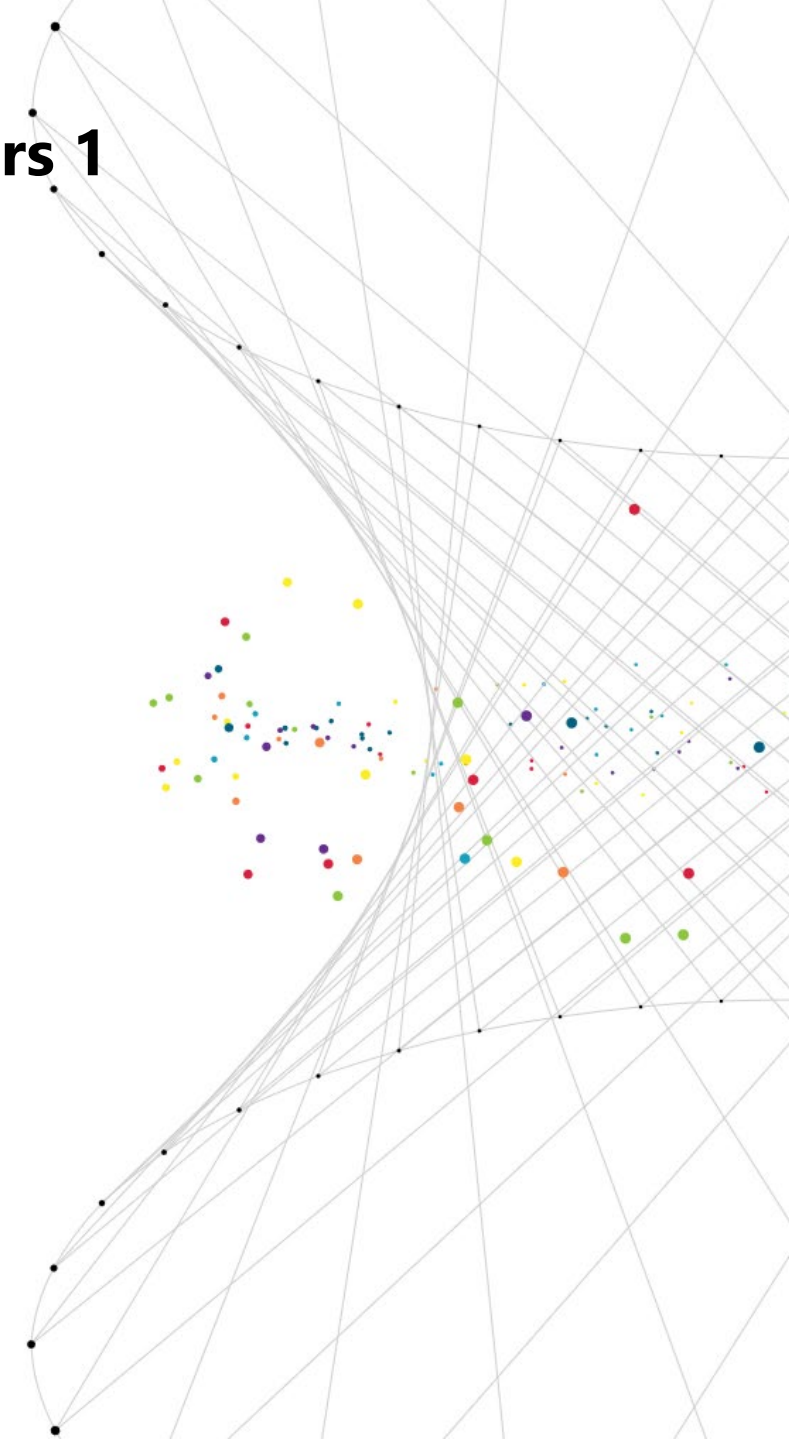
```
CREATE PROCEDURE DropStudentsinDeficit
AS
BEGIN
    declare @max int
    set @max = (select max(TakeID) from Take_S)

    declare @i int
    set @i = (select min(TakeID) from Take_S)

    WHILE (@i <= @max)
    BEGIN
        IF EXISTS (select @i from Take_S)
        BEGIN
            delete from Take_S
            where StudentID in (select ts.StudentID from TuitionStatus ts where TheStatus = 'Deficit')
            and TakeID = @i
            set @i = @i + 1
        END
    END
END
```



Financial System - Payment for Instructors 1

CalculateBalance: after insert trigger
Calculate balances of account for instructors

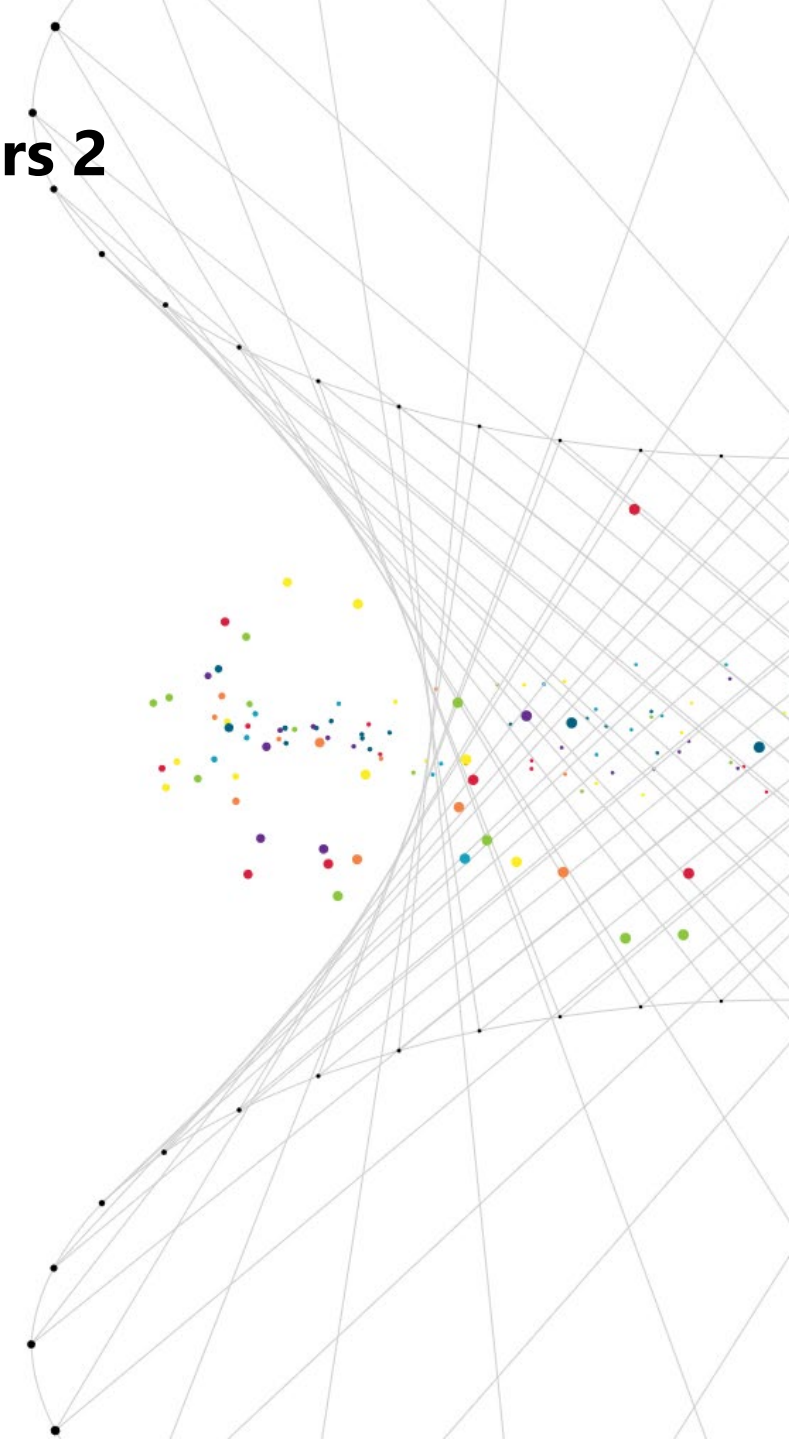
```
CREATE TRIGGER dbo.CalculateBalance
    ON dbo.Payment
    AFTER INSERT
AS
BEGIN

    Declare @Balance numeric(10,2)
    Declare @var numeric(10,2)

    set @var = (select TOP 1 Balance from Balance b join inserted i on b.InstructorID = i.InstructorID
                    Order by b.PaidTime DESC)
    set @Balance = (select TOP 1 Amount from inserted i Order BY i.PaidTime DESC) + isnull(@var, 0)


    insert into dbo.Balance
    select i.PaymentID, i.InstructorID, @Balance, i.PaidTime
    from inserted i

END
```



Financial System - Payment for Instructors 2

UpdateDepartmentBalance: after insert trigger
Update department balance after paying instructors

```
CREATE TRIGGER UpdateDepartmentBalance
    ON dbo.Payment
    AFTER INSERT
AS
BEGIN
    declare @Amount numeric(10,2)
    set @Amount = (select Amount from inserted)

    declare @Balance numeric(10,2)
    set @Balance = (select Balance from Department where DepartmentID =
                   (select i2.DepartmentID from inserted i1 join Instructor i2
                                                           on i1.InstructorID = i2.InstructorID))
    update Department
    set Balance = @Balance - @Amount
    where DepartmentID = (select i2.DepartmentID from inserted i1 join Instructor i2
                                                           on i1.InstructorID = i2.InstructorID)
END
```



Financial System - Payment for Instructors 3

SalaryPayment1: stored procedure
Pay a certain instructor

```
CREATE PROCEDURE SalaryPayment1 @InstructorID int, @quarters varchar(6), @years char(4)
AS
BEGIN
declare @Salary numeric(10,2)
declare @TeachFee numeric(10,2)
declare @TitleFee numeric(10,2)
declare @ServiceLength numeric(10,2)

set @TeachFee = isnull((select sum(c.TeachFee)...

if((select Title from Instructor
where InstructorID = @InstructorID) = 'Professor')
set @TitleFee = 50000
if((select Title from Instructor
 where InstructorID = @InstructorID) = 'AssociateProfessor')
set @TitleFee = 40000
if((select Title from Instructor
where InstructorID = @InstructorID) = 'AssistantProfessor')
set @TitleFee = 30000

declare @day datetime
set @day = (select EntryDate from Instructor
where InstructorID = @InstructorID)
if(DATEDIFF(year, @day, getdate()) < 5)...
if(DATEDIFF(year, @day, getdate()) < 10
and DATEDIFF(year, @day, getdate()) >= 5)
set @ServiceLength = 3000
if(DATEDIFF(year, @day, getdate()) < 15
```

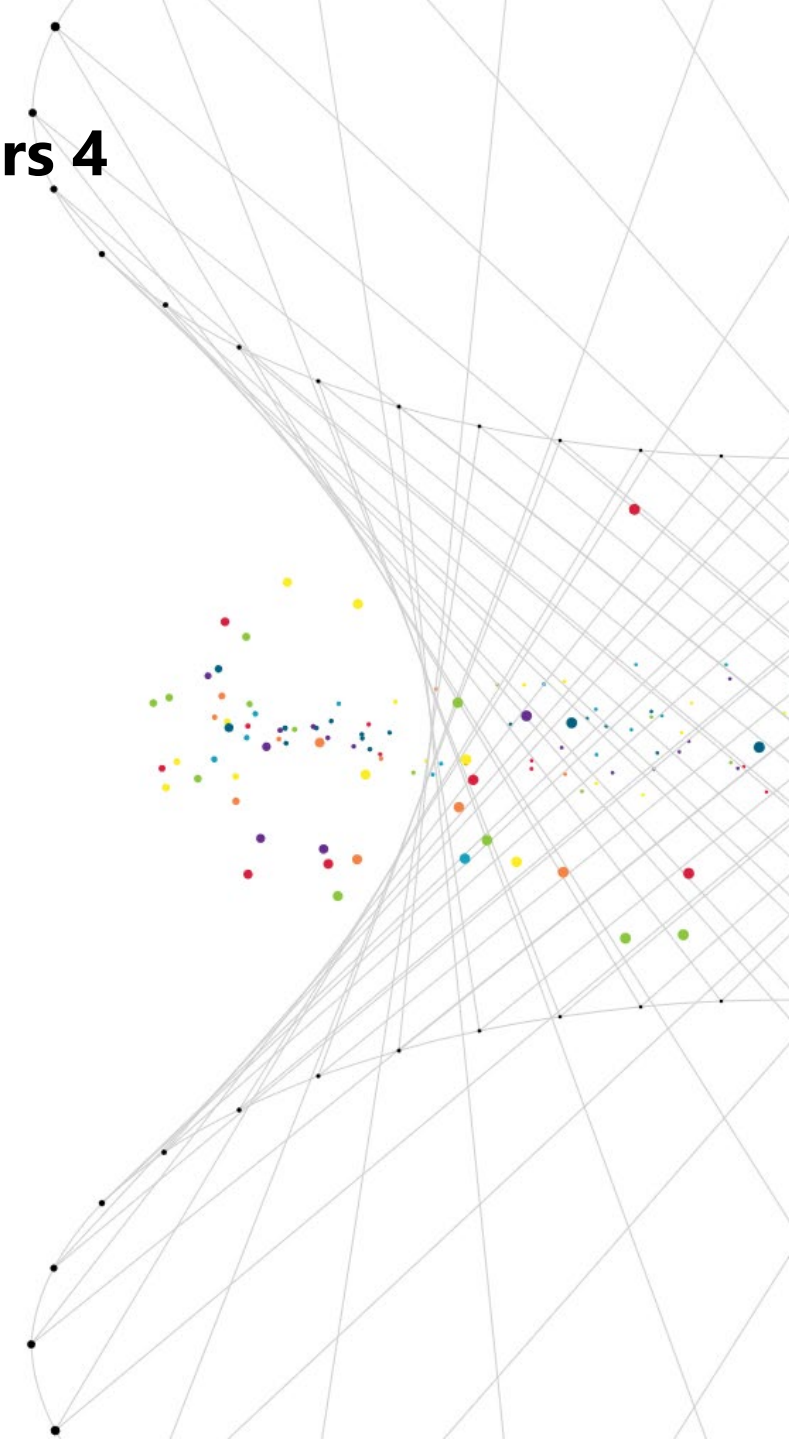
SalaryPaymentAll: stored procedure
Pay all instructors

```
CREATE PROCEDURE SalaryPaymentAll @quarters varchar(6), @years char(4)
AS
BEGIN
    declare @max int
    set @max = (select max(InstructorID) from Instructor)

    declare @i int
    set @i = (select min(InstructorID) from Instructor)
    WHILE (@i <= @max)
    BEGIN
        IF EXISTS (select @i from Instructor)
        BEGIN
            exec SalaryPayment1 @i, @quarters, @years
        END
        set @i = @i + 1
     END
END
```

Financial System - Payment for Instructors 4

InstructorPaymentReport: stored procedure
Input: ①first name, ②last name
Output: ①paid time, ②amount, ③balance of the account

```
CREATE PROCEDURE InstructorPaymentReport @FirstName nvarchar(20), @LastName nvarchar(20)
AS
BEGIN
select i.InstructorID, i.FirstName, i.LastName, p.Amount, b.Balance, p.PaidTime
from Payment p join Balance b on p.PaymentID = b.PaymentId
     join Instructor i on p.InstructorID = i.InstructorID
where i.FirstName = @FirstName and  i.LastName = @LastName
END

-- Test

--select * from Instructor
exec InstructorPaymentReport Hew, Probate
```



Data Flow Examples

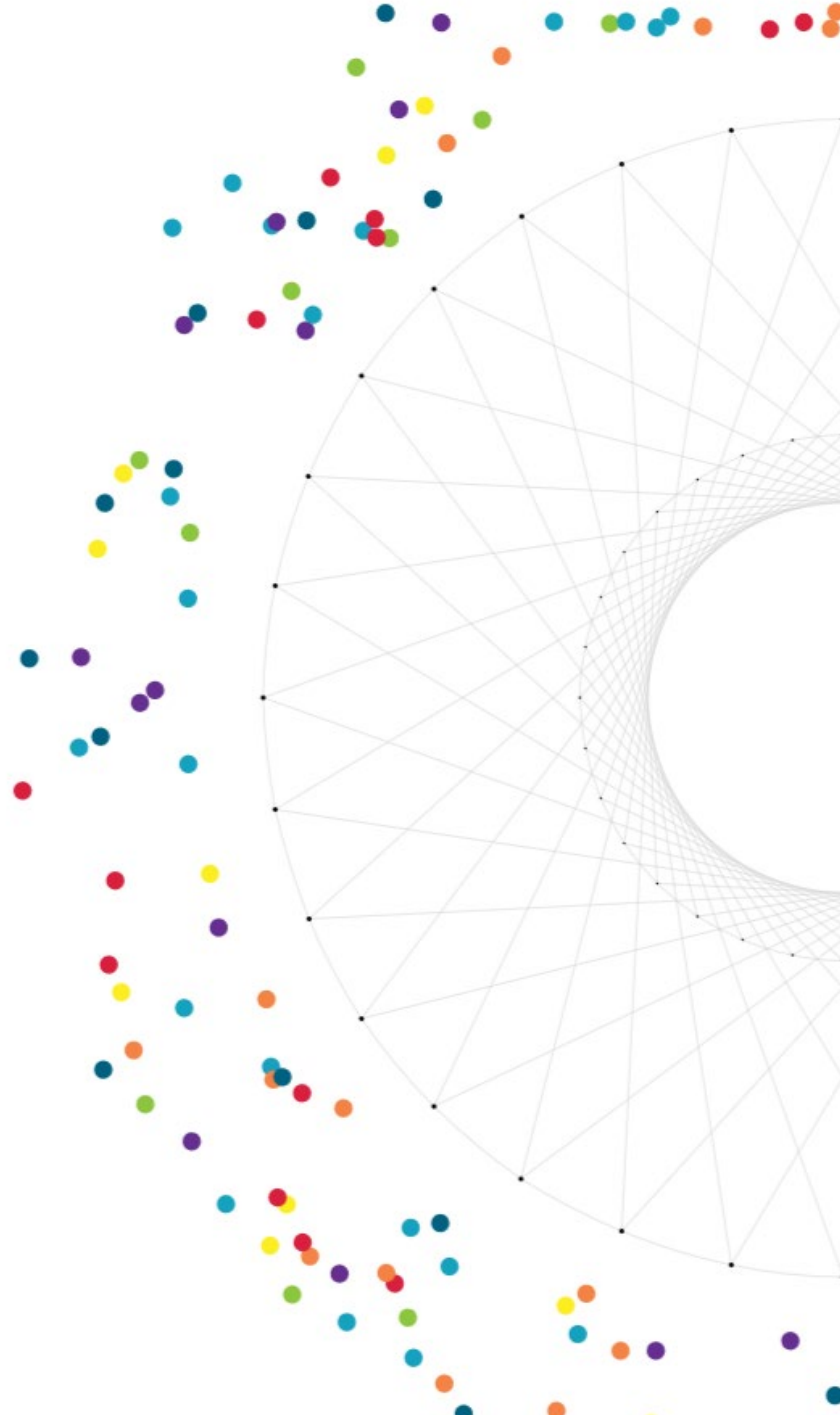
PART FOUR

Enrollment System

select * from **ListOfClasses2019**

Results | Messages

	TeachID	Course	Units	Department	Instructor	Locations	Term	CourseTime	Capacity	SpotsLeft
1	5	IntroMath	3	Science	Karisa Althrope	FranzHall 2744	Spring 2019	16:00:00.0000000	28	23
2	6	HistoryII	4	ArtandHistory	Vivianne Irons	FranzHall 3817	Summer 2019	08:00:00.0000000	43	37
3	7	IntroMath	3	Science	Katherina Torpie	FranzHall 3817	Fall 2019	12:00:00.0000000	34	27
4	8	LiteratureII	4	ArtandHistory	Vivianne Irons	MillerHall 2175	Fall 2019	10:00:00.0000000	29	24
5	10	IntroChemistry	3	Science	Cristionna Amoore	FranzHall 1854	Fall 2019	08:00:00.0000000	38	35
6	11	MediumBiology	4	Science	Karisa Althrope	MillerHall 3195	Winter 2019	14:00:00.0000000	50	43
7	12	EnglishI	3	Language	Brianna Harpur	FranzHall 1854	Fall 2019	12:00:00.0000000	30	24
8	13	EnglishI	3	Language	Arnoldo Steggals	FranzHall 3817	Winter 2019	14:00:00.0000000	34	25
9	15	Painting	5	ArtandHistory	York Mussared	KaplanHall 2493	Fall 2019	08:00:00.0000000	35	25
10	19	Germany	5	Language	Francesca Bilney	FranzHall 1854	Summer 2019	16:00:00.0000000	29	26
11	20	EnglishII	4	Language	Walliw de Chast...	MillerHall 3195	Spring 2019	14:00:00.0000000	28	20
12	22	EnglishIII	5	Language	Derick Czyz	FranzHall 2744	Fall 2019	14:00:00.0000000	32	27
13	23	Chinese	5	Language	Walliw de Chast...	KaplanHall 2493	Winter 2019	10:00:00.0000000	50	40
14	25	LiteratureI	3	ArtandHistory	Pierrette Green...	FranzHall 2744	Fall 2019	08:00:00.0000000	37	31



exec **Enrollment** 1, 4, 'Take' (no this class)

Results | Messages

	ErrorTime	Error
1	2019-06-02 19:34:26.117	No such courses in 2019!

	StudentID	TeachID	Course	Units	Department	Instructor	Locations	Term	CourseTime
1	1	5	IntroMath	3	Science	Karisa Althrope	FranzHall 2744	Spring 2019	16:00:00.0000000

exec **Enrollment** 1, 5, 'Take'
exec **Enrollment** 1, 5, 'Take' (repeated enrollment)

Results | Messages

	ErrorTime	Error
1	2019-06-02 19:35:45.503	Already enrolled!

	ErrorTime	Error
1	2019-06-02 19:35:45.503	Time Conlict!

	StudentID	TeachID	Course	Units	Department	Instructor	Locations	Term	CourseTime
1	1	5	IntroMath	3	Science	Karisa Althrope	FranzHall 2744	Spring 2019	16:00:00.0000000

exec **Enrollment** 1, 6, 'Take' (✓ this class / ✓ spots / X repeated / X time conflict)

Results | Messages

	StudentID	TeachID	Course	Units	Department	Instructor	Locations	Term	CourseTime
1	1	5	IntroMath	3	Science	Karisa Althrope	FranzHall 2744	Spring 2019	16:00:00.0000000
2	1	6	HistoryII	4	ArtandHistory	Vivianne Irons	FranzHall 3817	Summer 2019	08:00:00.0000000

exec **InstructorCoursesInfo** @InstructorID = 7, @Years = '2019', @Quarters =

Results | Messages

	InstructorID	InstructorName	CourseID	CourseName	Unit	CourseTime	DepartmentName	Locations
1	7	Katherina Torpie	1	IntroMath	3	12:00:00.0000000	Science	FranzHall 3817

exec **RollBook** @TeachID = 10

Results | Messages

	TakeID	StudentID	StudentName	TeachID	CourseName	HW1	HW2	HW3	HW4	Midterm1	Midterm2	Project1	Project2	Final	LetterGrade
1	6	52	Sasha Aberkirder	10	IntroChemistry	66.00	NULL	NULL	NULL	NULL	NULL	NULL	NULL	92.00	NULL
2	23	50	Erminia Mapis	10	IntroChemistry	94.70	NULL	NULL	NULL	NULL	NULL	NULL	NULL	72.60	NULL
3	37	43	Arvin Bagehot	10	IntroChemistry	79.80	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

exec **UpdateGrade_LG** 6, 'A'

Results | Messages

	TakeID	StudentID	StudentName	TeachID	CourseName	HW1	HW2	HW3	HW4	Midterm1	Midterm2	Project1	Project2	Final	LetterGrade
1	6	52	Sasha Aberkirder	10	IntroChemistry	66.00	NULL	NULL	NULL	NULL	NULL	NULL	NULL	92.00	A
2	23	50	Erminia Mapis	10	IntroChemistry	94.70	NULL	NULL	NULL	NULL	NULL	NULL	NULL	72.60	NULL
3	37	43	Arvin Bagehot	10	IntroChemistry	79.80	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Financial System

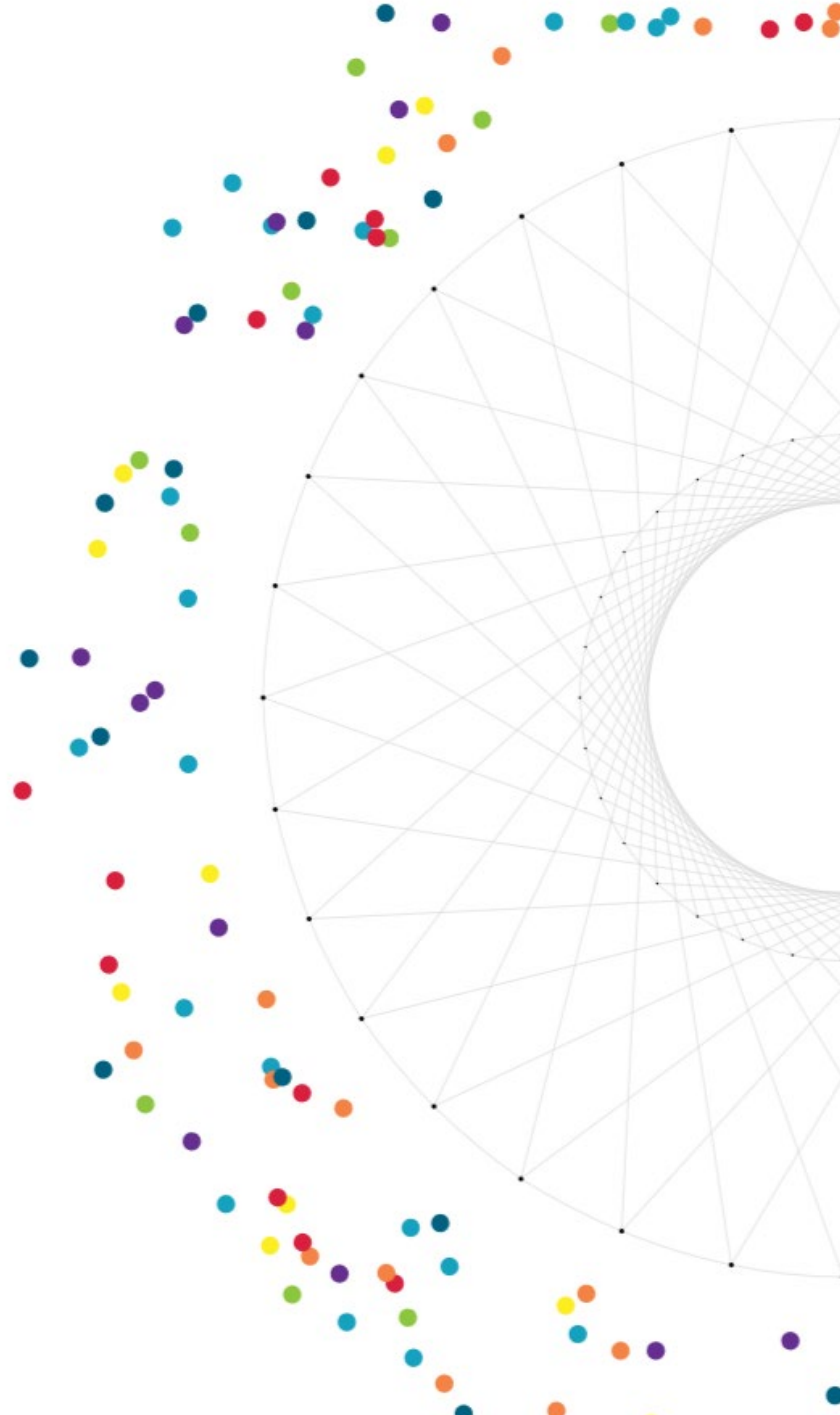
```
insert into Tuition values(1, 2000, getdate())
insert into Tuition values(2, 3000, getdate())
update Tuition set Amount = 3500 where TuitionID = 2007
delete from Tuition where TuitionID = 100
select * from Tuition
select * from TuitionAudit order by [Operation Time]
```

Results | Messages

	TuitionID	StudentID	Amount	PaidTime
101	2007	1	3500.00	2019-06-02 19:54:11.713
102	2008	2	3000.00	2019-06-02 19:54:15.933

Results | Messages

	TuitionID	StudentID	OldAmount	NewAmount	Operation	OperationTime
101	2007	1	NULL	2000.00	insert	2019-06-02 19:54:11.713
102	2008	2	NULL	3000.00	insert	2019-06-02 19:54:15.933
103	2007	1	2000.00	3500.00	update	2019-06-02 19:56:23.823
104	100	94	5954.06	NULL	delete	2019-06-02 19:59:15.413



exec **TuitionCheckAll**

Results | Messages

	StudentID	Paid	Payable	TheStatus
1	1	3907.61	1000.00	Surplus
1	2	5472.64	0.00	Surplus
1	3	3239.39	1000.00	Surplus
1	4	3155.49	3750.00	Deficit
1	5	2687.47	0.00	Surplus
1	6	3669.47	0.00	Surplus
1	7	3143.24	0.00	Surplus
1	8	3848.73	3000.00	Surplus

...

Results | Messages

	StudentID	Paid	Payable	TheStatus
1	93	3965.70	0.00	Surplus
1	94	5954.06	0.00	Surplus
1	95	2286.64	1500.00	Surplus
1	96	4402.07	1500.00	Surplus
1	97	2046.91	1000.00	Surplus
1	98	5983.31	1500.00	Surplus
1	99	5827.67	0.00	Surplus
1	100	3252.70	1500.00	Surplus

select * from Take_S
exec **DropStudentsinDeficit**
select * from Take_S

Results | Messages

	TakeID	StudentID	TeachID	HW1	HW2	HW3	HW4
30	34	90	19	72.20	NULL	NULL	NULL
31	35	4	11	66.00	NULL	NULL	NULL
32	36	15	6	82.90	NULL	NULL	NULL
33	37	43	10	79.80	NULL	NULL	NULL
34	38	66	13	86.80	NULL	NULL	NULL
35	39	79	12	60.20	NULL	NULL	NULL

Results | Messages

	TakeID	StudentID	TeachID	HW1	HW2	HW3	HW4
21	29	69	5	71.60	NULL	NULL	NULL
22	31	15	22	80.20	NULL	NULL	NULL
23	32	1	5	96.80	NULL	NULL	NULL
24	34	90	19	72.20	NULL	NULL	NULL
25	36	15	6	82.90	NULL	NULL	NULL
26	38	66	13	86.80	NULL	NULL	NULL

exec **SalaryPaymentAll** 'Spring', '2019'
select * from Payment
exec **InstructorPaymentReport** 'Karisa', 'Althrope'
select * from Department

Payment Table

Results | Messages

	PaymentID	InstructorID	Amount	PaidTime
21	1061	1	42000....	2019-06-02 20:23:40.390
22	1062	2	33000....	2019-06-02 20:23:40.390
23	1063	3	44000....	2019-06-02 20:23:40.390
24	1064	4	55000....	2019-06-02 20:23:40.390
25	1065	5	52000....	2019-06-02 20:23:40.390
26	1066	6	54000....	2019-06-02 20:23:40.390
27	1067	7	33000....	2019-06-02 20:23:40.390
28	1068	8	33000....	2019-06-02 20:23:40.390
29	1069	9	42000....	2019-06-02 20:23:40.390
30	1070	10	55000....	2019-06-02 20:23:40.390
31	1071	11	54000....	2019-06-02 20:23:40.390
32	1072	12	54000....	2019-06-02 20:23:40.390
33	1073	13	42000....	2019-06-02 20:23:40.390
34	1074	14	44750....	2019-06-02 20:23:40.390
35	1075	15	45000....	2019-06-02 20:23:40.390
36	1076	16	44000....	2019-06-02 20:23:40.390
37	1077	17	33500....	2019-06-02 20:23:40.390
38	1078	18	55000....	2019-06-02 20:23:40.390
39	1079	19	55000....	2019-06-02 20:23:40.390
40	1080	20	44000....	2019-06-02 20:23:40.390

Payment Report for Instructor Karisa Althrope

Results | Messages

	PaidTime	FirstName	LastName	Amount	Balance
1	2019-03-11 19:21:05.000	Karisa	Althrope	8026.26	8026.26
2	2019-06-02 20:23:40.390	Karisa	Althrope	33500.00	41526.26

Department Table before Payment

Results | Messages

	DepartmentID	DepartmentName	Building	Budget	Balance
1	1	Science	SceienceBuilding	721272.00	3075048.00
2	2	Language	LanguageCenter	837979.00	1494114.00
3	3	ArtandHistory	SocialScienceHall	127751.00	3938616.00

Department Table after Payment

Results | Messages

	DepartmentID	DepartmentName	Building	Budget	Balance
1	1	Science	SceienceBuilding	721272.00	2836548.00
2	2	Language	LanguageCenter	837979.00	1113364.00
3	3	ArtandHistory	SocialScienceHall	127751.00	3643616.00

Report Examples

PART FIVE

EnrolledCoursesInfo: View

Provide students with information of enrolled courses

```
create view EnrolledCoursesInfo
AS
select ts.StudentID, ts.TakeID, ti.TeachID,
       s.FirstName +' '+ s.LastName as StudentName,
       c.CourseID, c.CourseName, c.Unit, d.DepartmentName,
       cl.Building+ ' ' + cl.RoomNo as Locations,
       ti.Quarters + ' ' + ti.Years as Term, ti.CourseTime as CourseTime
from Course as c
inner join Teach_I as ti on c.CourseID = ti.CourseID
inner join Department as d on c.DepartmentID = d.DepartmentID
inner join Classroom as cl on cl.ClassroomID = ti.ClassroomID
inner join Take_S as ts on ti.TeachID = ts.TeachID
inner join Student as s on ts.StudentID = s.StudentID;
```

select * from **EnrolledCoursesInfo** where StudentID = 56 and Term = 'Fall 2019'

Results | Messages

	StudentID	TakeID	TeachID	StudentName	CourseID	CourseName	Unit	DepartmentName	Locations	Term	CourseTime
1	56	15	7	Ruggiero Gibbetts	1	IntroMath	3	Science	FranzHall 3817	Fall 2019	12:00:00.0000000

GradeView: stored procedure

Help students to view their grades

```
CREATE PROCEDURE GradeView @StudentID int
AS
BEGIN
select ts.TakeID as TakeID, s.StudentID as StudentID,
       s.FirstName +' '+ s.LastName as StudentName,
       ti.TeachID as TeachID, c.CourseName as CourseName,
       ts.HW1 as HW1, ts.HW2 as HW2, ts.HW3 as HW3,
       ts.HW4 as HW4, ts.Midterm1 as Midterm1,
       ts.Midterm2 as Midterm2, ts.Project1 as Project1,
       ts.Project2 as Project2,
       ts.Final as Final, ts.LetterGrade as LetterGrade
from Take_S ts join Teach_I ti on ts.TeachID = ti.TeachID
     join Student s on ts.StudentID = s.StudentID
     join Course c on ti.CourseID = c.CourseID
where ts.StudentID = @StudentID
END
```

exec **GradeView** 56

Results | Messages

	TakeID	StudentID	StudentName	TeachID	CourseName	HW1	HW2	HW3	HW4	Midterm1	Midterm2	Project1	Project2	Final	LetterGrade
1	15	56	Ruggiero Gibbetts	7	IntroMath	86.60	NULL	NULL	NULL	NULL	NULL	NULL	NULL	95.30	NULL
2	16	56	Ruggiero Gibbetts	20	EnglishII	93.10	NULL	NULL	NULL	NULL	NULL	NULL	NULL	95.00	NULL

Get Top5 highest average final exam score class

```
Select top(5) ti.TeachID, ti.CourseID, c.CourseName, avg(ts.Final) as [Average Final]
from Teach_I as ti
join Course as c on ti.CourseID = c.CourseID
join Take_S as ts on ti.TeachID = ts.TeachID
group by ti.TeachID, ti.CourseID, c.CourseName
order by avg(ts.Final) desc;
```

Results | Messages

	TeachID	CourseID	CourseName	Average Final
1	15	26	Painting	83.390000
2	20	14	EnglishII	82.850000
3	11	11	MediumBiology	82.471428
4	10	7	IntroChemistry	82.300000
5	13	13	EnglishI	81.500000

Get Top5 popular courses in 2018 and 2019

```sql
Select top(5) ti.CourseID, c.CourseName, count(StudentID) as [Number of Enrolled Students]
from Teach_I as ti
inner join Course as c on ti.CourseID = c.CourseID
left join Take_S as ts on ti.TeachID = ts.TeachID
group by ti.CourseID, c.CourseName
order by count(StudentID) desc;
```

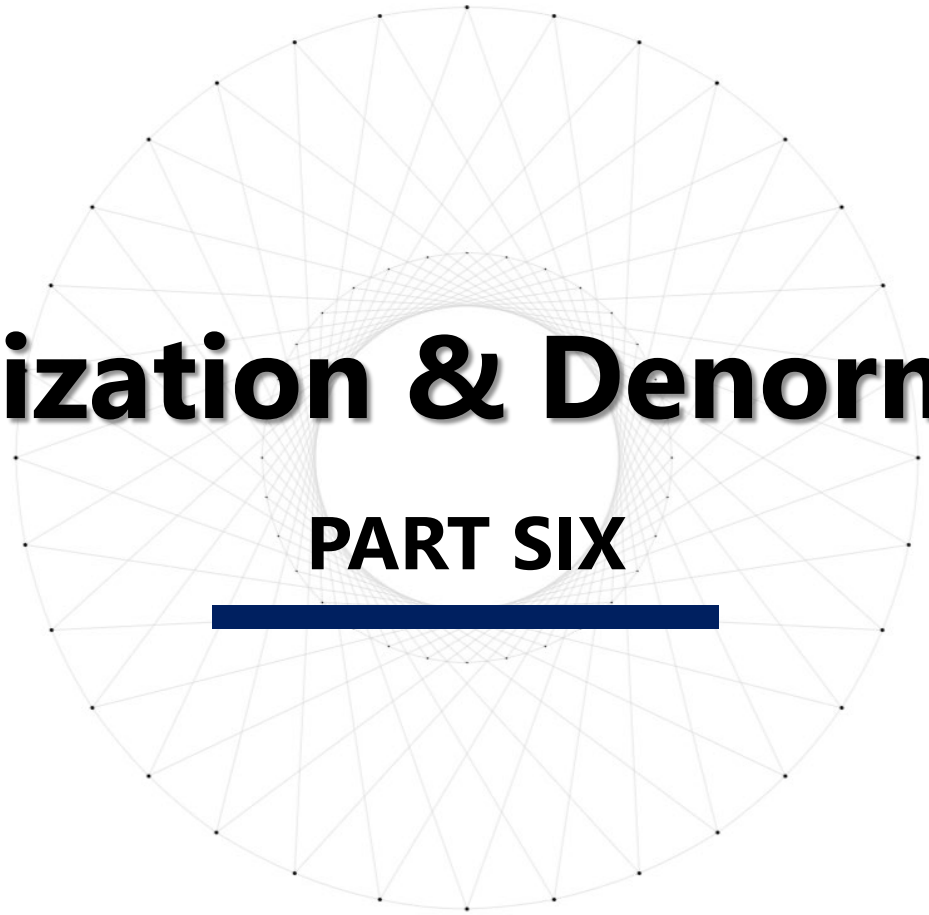
Results | Messages

	CourseID	CourseName	Number of Enrolled Students
1	13	EnglishI	15
2	1	IntroMath	12
3	16	Chinese	10
4	26	Painting	10
5	14	EnglishII	8

Get instructors' latest account info in the order of account balance (max → min)

```sql
select a.InstructorID , b.balance, a.[Latest PaidTime]
from (select InstructorID, MAX(PaidTime) as [Latest PaidTime]
      from Balance group by InstructorID) a
join balance b on  a.InstructorID = b.InstructorID
where a.[Latest PaidTime] = b.PaidTime
order by b.balance desc;
```

	InstructorID	balance	Latest PaidTime
1	13	8562.09	2019-04-07 07:44:22.000
2	3	8399.36	2019-03-27 00:31:38.000
3	17	8026.26	2019-03-11 19:21:05.000
4	2	7724.09	2019-03-19 12:28:32.000
5	5	6769.33	2019-03-09 11:15:20.000
6	1	3965.11	2019-03-15 13:08:14.000
7	16	2324.76	2019-04-24 08:31:07.000
8	11	2179.95	2019-03-13 01:41:32.000
9	10	2070.45	2019-05-24 19:07:55.000
10	9	975.64	2019-05-07 05:50:22.000
11	15	730.49	2019-04-29 13:41:43.000
12	19	180.79	2019-04-13 23:16:58.000
13	20	-4.53	2019-03-17 18:50:37.000
14	14	-1728....	2019-05-19 07:39:32.000
15	7	-1778....	2019-05-02 23:23:03.000
16	6	-4085....	2019-04-13 22:46:38.000
17	8	-4597....	2019-05-23 05:00:33.000
18	12	-5323....	2019-04-26 05:45:20.000
19	4	-8036....	2019-03-30 01:47:03.000
20	18	-8433....	2019-05-15 19:14:03.000



Normalization & Denormalization

PART SIX

1st Normal Form: No repeated columns

2nd Normal Form: Dependence on Primary Key

3rd Normal Form: No transitive dependence on Primary Key

Security & Audit/Logging

PART SEVEN

- **Tuition Audit table**
- **Authorization of stored procedures and views**

Stored Procedures	Users
Enrollment	Students
GradeView	Students
UpdateGrade	Instructors
RollBook	Instructors
InstructorPaymentReport	Instructors, HR
InstructorCoursesInfo	Instructors
TuitionCheck1, TuitionCheckAll	Student Financial Staff
DropStudentsinDeficit	Student Financial Staff
SalaryPayment1, SalaryPaymentAll	HR
Views	**Users**
ListOfClasses2019	All
EnrolledcoursesInfo	Students

Performance Improvement

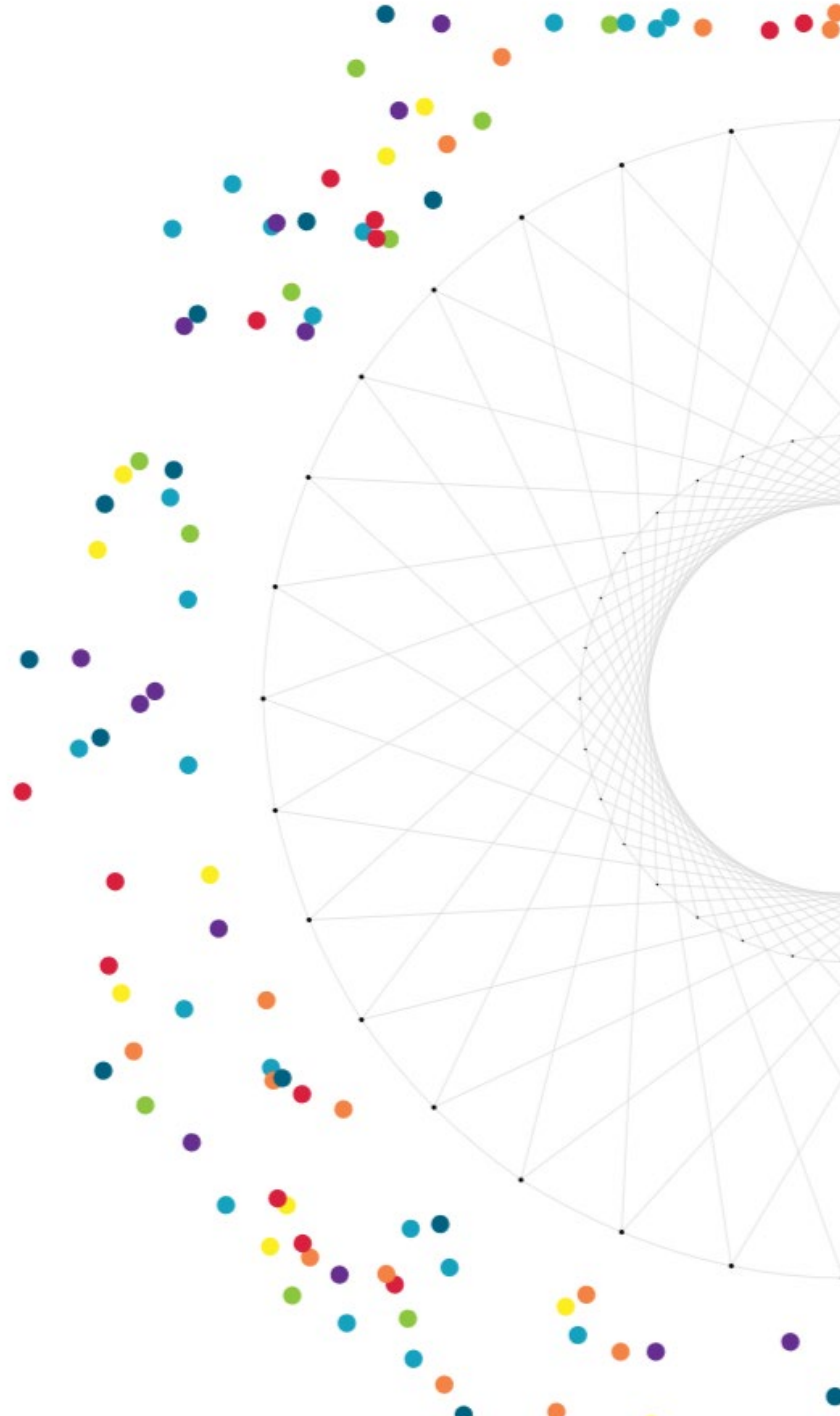
PART EIGHT

- **Clustered Index**

  create clustered index on tables when the same columns are heavily used

- **Stored Procedure**

- **Denormalization**



Performance & Storage Assessment

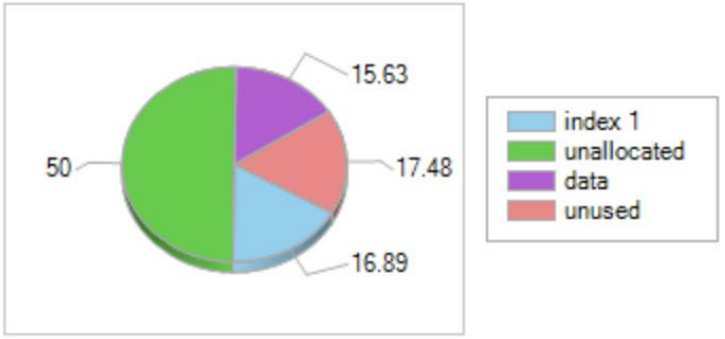
PART NINE

Disk Usage
[Project-SchoolManagement]
on EB876962BAC3 at 6/4/2019 4:57:17 PM

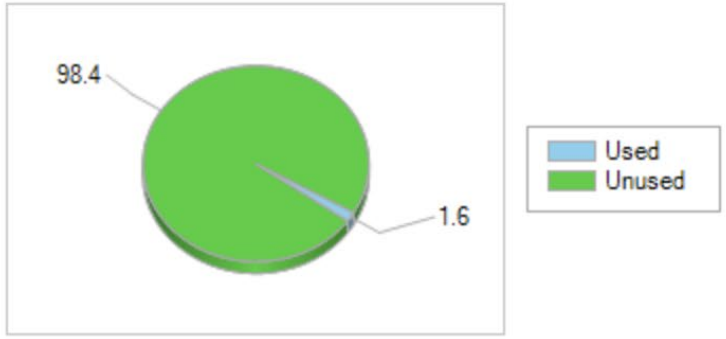
This report provides overview of the utilization of disk space within the Database.

Total Space Reserved	80.00 MB
Data Files Space Reserved	8.00 MB
Transaction Log Space Reserved	72.00 MB

Data Files Space Usage (%)



Transaction Log Space Usage (%)



No entry found for autogrow/autoshrink event for Project-SchoolManagement database in the trace log.

Disk Space Used by Data Files

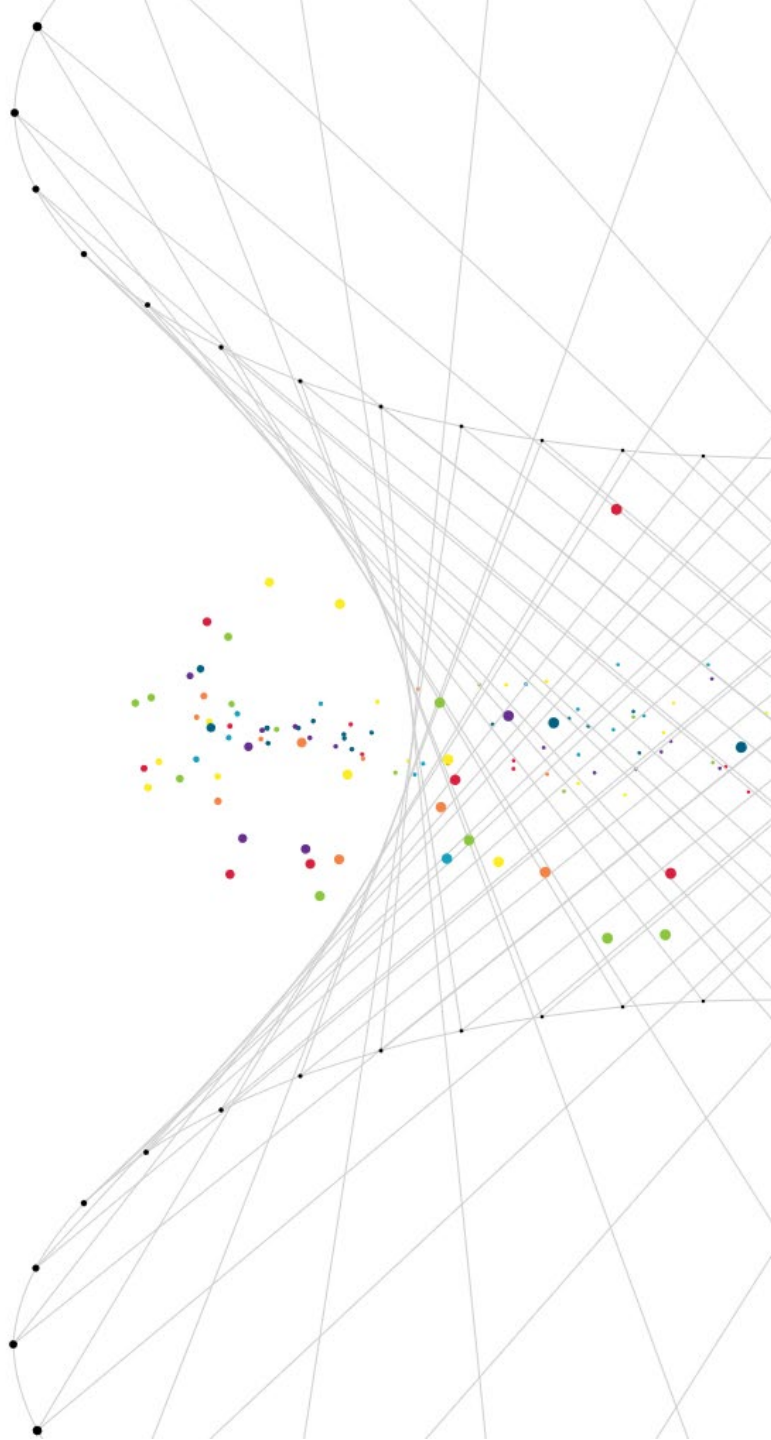
Filegroup Name	Logical File Name	Physical File Name	Space Reserved	Space Used
PRIMARY	Project-SchoolManagement	C:\DB\Project-SchoolManagement.mdf	8.00 MB	4.13 MB

Disk Usage by Top Tables
[Project-SchoolManagement]
SQL Server

on EB876962BAC3 at 6/4/2019 4:58:20 PM

This report provides detailed data on the utilization of disk space by top 1000 tables within the Database. The report does not provide data for memory optimized tables.

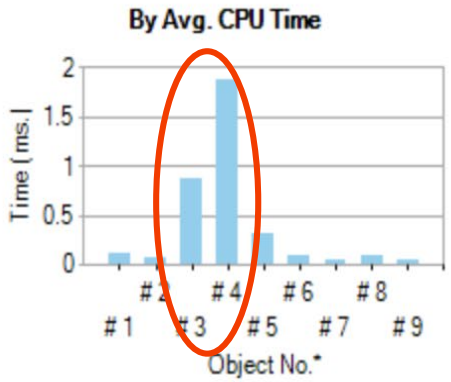
Table Name	# Records	Reserved (KB)	Data (KB)	Indexes (KB)	Unused (KB)
dbo.Tuition	100	72	8	8	56
dbo.TuitionStatus	100	72	8	8	56
dbo.TuitionAudit	0	72	8	8	56
dbo.Department	3	72	8	8	56
dbo.Instructor	20	72	8	8	56
dbo.Student	100	72	16	16	40
dbo.Course	27	72	8	8	56
dbo.Classroom	8	72	8	8	56
dbo.Teach_I	22	72	8	8	56
dbo.DynEnroll	14	72	8	8	56
dbo.Take_S	42	72	8	8	56
dbo.Payment	20	72	8	8	56
dbo.Balance	20	72	8	8	56

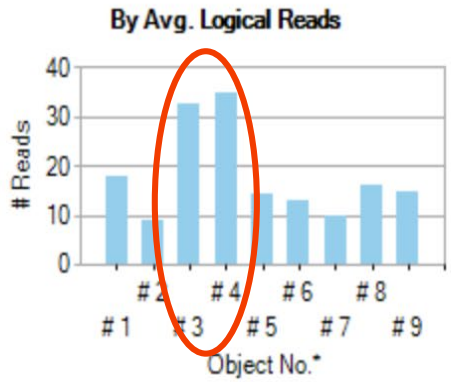


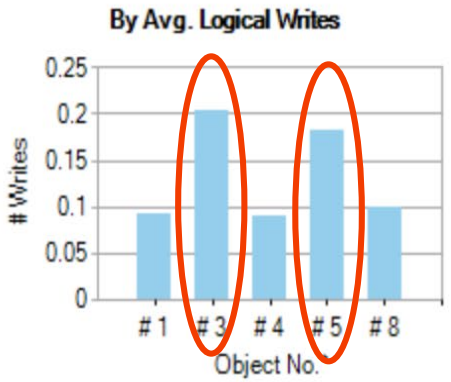
Object Execution Statistics
[Project-SchoolManagement]
SQL Server

on EB876962BAC3 at 6/4/2019 4:58:54 PM

Top Executable Objects





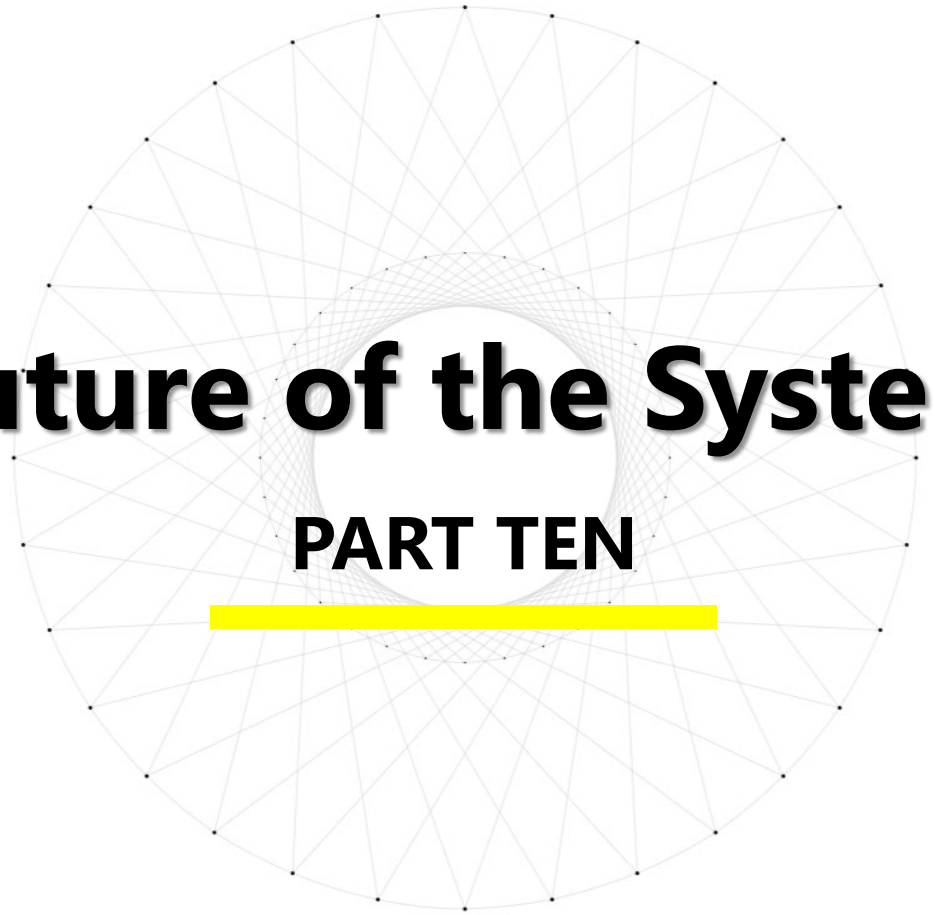


Object No.*	Object Name	Object Type	Avg. CPU Time (ms.)	Total CPU Time (%)	# Avg. Logical Reads	# Avg. Logical Writes	# Avg. Logical IO	Total Logical IO (%)
1	[dbo]..TeachCheck	SQL Trigger	0.11	1.85	18.09	0.09	18.18	8.42
2	[dbo]..ChangeSpotsLeft	SQL Trigger	0.07	1.10	9.00	0.00	9.00	4.06
3	[dbo]..EnrollCheck	SQL Trigger	0.87	23.88	32.62	0.20	32.82	23.57
4	[dbo]..Enrollment	SQL Stored-Procedure	1.88	64.89	35.02	0.09	35.10	36.17
5	[dbo]..UpdateGrade_HW1	SQL Stored-Procedure	0.32	4.84	14.16	0.18	14.34	6.48
6	[dbo]..UpdateGrade_FL	SQL Stored-Procedure	0.09	1.32	13.02	0.00	13.02	5.88
7	[dbo]..ShowUpadatedGrade	SQL Trigger	0.04	1.20	10.00	0.00	10.00	9.03
8	[dbo]..CalculateBalance	SQL Trigger	0.09	0.62	16.20	0.10	16.30	3.31
9	[dbo]..UpdateDepartmentBalance	SQL Trigger	0.04	0.30	15.00	0.00	15.00	3.08

3	[dbo]..EnrollCheck	SQL Trigger		0.87	23.88	32.62	0.20	32.82	23.57

SQL Statement	# Executions (With Last Plan)	# Plans Generated	Avg. CPU Time (ms.)		# Avg. Logical Reads	# Avg. Logical Writes	# Avg. Logical IO	
set @num3 = (select count(*) from inserted i join ListOfClasses2019 lc on i.TeachID = lc.TeachID	101	1	0.03		2.93	0.00	2.93	
set @dyn = (select SpotsLeft from DynEnroll where TeachID = (select TeachID from inserted)	101	1	0.02		2.00	0.00	2.00	
set @num2 = (select count(*) from Take_S ts join inserted i on ts.TeachID = i.TeachID and ts.StudentID = i.StudentID	98	2	0.02		2.00	0.00	2.00	
set @num1 = (select count(*) from Take_S ts join ListOfClasses2019 lc on ts.TeachID = lc.TeachID where ts.StudentID = (select StudentID from inserted) and lc.CourseTime = (select lc1.CourseTime from inserted i1 join ListOfClasses2019 lc1 on i1.TeachID = lc1.TeachID) and lc.Term = (select lc2.Term from inserted i2 join ListOfClasses2019 lc2 on i2.TeachID = lc2.TeachID)	98	3	0.49		8.57	0.00	8.57	
insert into Take_S(StudentID, TeachID) select StudentID, TeachID from inserte	44	1	0.30		17.11	0.20	17.32	

4	[dbo]..Enrollment	SQL Stored-Procedure		1.88	64.89	35.02	0.09	35.10	36.17
	SQL Statement	**# Executions (With Last Plan)**	**# Plans Generated**	**Avg. CPU Time (ms.)**		**# Avg. Logical Reads**	**# Avg. Logical Writes**	**# Avg. Logical IO**	
	insert into Take_S(StudentID, TeachID, LetterGrade) values (@StudentID, @TeachID, 'NA'	101	1	1.28		24.23	0.09	24.32	
	select StudentID, ts.TeachID as TeachID, Course, Units, Department, Instructor, Locations, Term, CourseTime from Take_S ts join ListOfClasses2019 lc on ts.TeachID = lc.TeachID where StudentID = @StudentI	99	2	0.60		10.79	0.00	10.79	
5	[dbo]..UpdateGrade_HW1	SQL Stored-Procedure		0.32	4.84	14.16	0.18	14.34	6.48
	SQL Statement	**# Executions (With Last Plan)**	**# Plans Generated**	**Avg. CPU Time (ms.)**		**# Avg. Logical Reads**	**# Avg. Logical Writes**	**# Avg. Logical IO**	
	update Take_S set HW1 = @HW1 where TakeID = @TakeI	44	1	0.32		14.16	0.18	14.34	



Future of the System

PART TEN

- Create accounts for different users (students, instructors, staffs, etc.)
- Limit accounts' access to views, stored procedures and other tables.
- Update budgets and balances of departments every year.
- Set prerequisites for each course.
- Warning signs for students who fail or do not take enough courses.
- Develop reservation systems of rooms.
- Connect different database systems, such as library systems, dining hall systems, gym systems, etc.


THANK YOU