5.0 Integrity Constraints

- Integrity constraints are those constraints in database system which guard against invalid database operations or accidental damage to the database, by ensuring that authorized changes to the database. It does not allow to loss of data consistency in database, it ensures database consistency.
- In fact, integrity constraints provide a way of ensuring that changes made to the database by authorized users do not result in a loss of data consistency.
- Example of integrity constraints in E-R model
  - Key declaration: candidate key, primary key
  - Form of relationship : mapping cardinalities: one to one, one to many etc
- In database management system we can enforce any arbitrary predicate as integrity constraints but it adds overhead to the database system so its cost should be evaluated, as far as possible integrity constraint should with minimal overhead.

5.1 Domain Constraints

- Set of all possible values for attribute known as its domain. Domain constraints enforce attribute should hold only particular types of attributes. A domain of possible values should be associated with every attribute. Domain constraints are the most elementary form of integrity constraint. It is tested by database system whenever a new data item is entered into database. System test values inserted in the database and test queries to ensure that the comparisons make sense.

Domain types in SQL

SQL standard supports a variety of built in domain types including:

- Char (n): A fixed length character string with user specified length n.
- Varchar(n): A variable length string with user specified maximum length n.
- Int: An integer (Machine dependant).
- Smallint: A small integer.
- Numeric (p,d): A fixed point number with user specified precision. Where (.) is counted in p.
- Real, double precision: Floating point and double precision floating point numbers.
- Float (n): A floating point number with precision of at least n digits.
- Date: A calendar date containing a four digit year, month and day of the month.
- Time: The time of a day, in hours, minutes and seconds.
- Timestamp: A combination of date and time.

- New domains can be created from existing data types
- E.g. **create domain** *Dollars* **numeric**(12, 2)
  create domain *Pounds* **numeric**(12,2)
- The **check** clause in SQL allow domains to be restricted
  Example 1

  create domain *salary-rate* **numeric(5)**
  constraint *value-test* **check**(*value* > = 5000)

  The domain constraint ensures that the hourly-rate must greater than 5000

  The clause **constraint** *value-test is* optional but useful to indicate which constraint an update violated.

Example 2:

- **create domain** *AccountType* **char**(10)
  constraint *account-type-test*
  check (**value in** ('Checking', 'Saving'))

Example 3:

create domain account-number **char(10)**
constraint account-number-null-test **check**(value not null)

5.2 Referential Integrity

- Referential integrity is a condition which Ensures that a value that appears in one relation for a given set of attributes also appears for a certain set of attributes in another relation.

Example

If "B1" is a branch name appearing in one of the tuples in the *account* relation, then there exists a tuple in the *branch* relation where "B1" exist for branch name attribute.
.

Example:
Consider two relation department and employee as follows
department(deptno#,dname)
employee(empno#,ename,deptno)

- Deletion of particular department from department table also need to delete records of employees they belongs to that particular department or delete need not be allow if there is any employee that is associated to that particular department that we are going to delete.
- Any update made in deptno in department table deptno in employee must be updated automatically.
- This implies primary key acts as a referential integrity constraint in a relation.

Formal Definition

- Let $r_1(R_1)$ and $r_2(R_2)$ be relations with primary keys K_1 and K_2 respectively. The subset α of R_2 is a ***foreign key*** referencing K_1 in relation r_1, if for every t_2 in r_2 there must be a tuple t_1 in $r1$ such that $t_1[K_1] = t_2[\alpha]$.
- Referential integrity constraint also called subset dependency since its can be written as
  $\prod_{\alpha} (r_2) \subseteq \prod K_1 (r_1)$

5.2.1 Referential integrity in E-R Model

- Consider relationship set R between entity sets E_1 and E_2. The relational schema for R includes the primary keys K_1 of E_1 and K_2 of E_2. Then K_1 and K_2 form foreign keys on the relational schemas for E_1 and E_2 respectively that leads referential integrity constraint.

- Weak entity sets are also a source of referential integrity constraints. A weak entity set must include the primary key attributes of the entity set on which it depends

5.2.2 Database modification

- The following tests must be made in order to preserve the following referential integrity constraint:
  $\prod_{\alpha} (r_2) \subseteq \prod_{K} (r_1)$
- Insert. If a tuple t_2 is inserted into r_2, the system must ensure that there is a tuple t_1 in r_1 such that $t_1[K] = t_2[\alpha]$. That is
  $t_2 [\alpha] \in \prod_{K} (r_1)$
- Delete. If a tuple, t_1 is deleted from r_1, the system must compute the set of tuples in r_2 that reference $t1$:
  $\sigma_{\alpha} = t_1[K] (r_2)$
- If this set is not empty
  - either the delete command is rejected as an error, or
  - the tuples that reference t_1 must themselves be deleted (cascading deletions are possible).

- Update: There are two cases:
  - If a tuple t_2 is updated in relation r_2 and the update modifies values for foreign key α, then a test similar to the insert case is made:
    - Let t_2' denote the new value of tuple t_2. The system must ensure that $t_2'[\alpha] \in \prod K(r_1)$
  - If a tuple t_1 is updated in r_1, and the update modifies values for the primary key (K), then a test similar to the delete case is made:
    1. The system must compute
       $\sigma\alpha = t_1[K] (r_2)$
       using the old value of t_1 (the value before the update is applied).
    2. If this set is not empty
       1. the update may be rejected as an error, or
       2. the update may be cascaded to the tuples in the set, or
       3. the tuples in the set may be deleted.

5.2.3 Referential integrity in SQL

- Using the SQL Create table statement we can enforce
  - Primary key
  - Unique.
  - Foreign key

Example:

```
create table customer
(
  customer-name     char(20),
  customer-street   char(30),
  customer-city     char(30),
  primary key (customer-name)
)

create table branch
(
   branch-name      char(15),
   branch-city char(30),
   assets       integer,
 primary key (branch-name)
)
```

```
create table account
(
  account-number     char(10),
  branch-name        char(15),
  balance       integer,
  primary key (account-number),
  foreign key (branch-name) references branch
)

create table depositor
(
  customer-name      char(20),
  account-number     char(10),
  primary key (customer-name, account-number),
  foreign key (account-number) references account,
  foreign key (customer-name) references customer
)
```

Cascading actions

Syntax

```
create table account
        . . .
    foreign key(branch-name) references branch
                on delete cascade
            on update cascade
    . . . )
```

on delete cascade: if a delete of a tuple in *branch* results referential-integrity constraint violation, it also delete tuples in relation account that refers to the branch that was deleted.

on update cascade : if a update of a tuple in *branch* results referential-integrity constraint violation, it updates tuples in relation account that refers to the branch that was updated.

5.3 Assertion

- An assertion is a predicate expressing a condition we wish the database to always satisfy
- Domain constraints, functional dependency and referential integrity are special forms of assertion.
- If a constraint cannot be expressed in these forms, we use an assertion
- e.g.
  - Sum of loan amounts for each branch is less than the sum of all account balances at the branch.
  - Every loan customer keeps a minimum of $1000 in an account.
- General syntax for creating assertion in SQL is
  create assertion <assertion-name> **check** <predicate>

- Example 1: sum of loan amounts for each branch is less than the sum of all account balances at the branch.

```
create assertion sum-constraint check
    (not exists (select * from branch
        where (select sum(amount) from loan
            where loan.branch-name = branch.branch-name)
        >= (select sum(amount) from account
            where loan.branch-name = branch.branch-name)))
```

Example 2: every customer must have minimum balance 1000 in an account who are loan holder

```
create assertion balance-constraint check
   (not exists (
      select * from loan
          where not exists (
              select *from borrower, depositor, account
                where loan.loan-number = borrower.loan-number
                    and borrower.customer-name = depositor.customer-name
                    and depositor.account-number = account.account-number
                    and account.balance >= 1000)))
```

- When an assertion is created, the system tests it for validity. If the assertion is valid then only allow further modification. if test found assertion is violated then it can not go ahead.
- Assertion testing may introduce a significant amount of overhead, especially if the assertions are complex; hence assertions should be used with great care.

5.4 Trigger

A trigger is a statement that is automatically executed by the system as a side effect of a modification to
the database. While writing a trigger we must specify

- conditions under which the trigger is executed
- actions to be taken when trigger executes

Triggers re useful mechanism to perform certain task automatically when certain condition/s met. Sometime trigger is also called rule or action rule.

Basic syntax for trigger

```
CREATE OR REPLACE TRIGGER <TRIGGER NAMR>
{BEFORE,AFTER}
    {INSERT|DELETE|UPDATE [OF column, . .]} ON <table name>
[REFERENCING {OLD AS <old>, NEW AS <new>}]
[FOR EACH ROW [WHEN <condition>]]
DECLARE
   Variable declaration;
BEGIN
 . . .
END;
```

5.0 Integrity Constraints

Example 1: maintaining log

emp(empno,ename,sal)
emp_log(empno,ename,sal,operation_perform,userid,opr_date

```
Create or replace emp_operation_log
After update or delete on emp
For each row
Declare
   oper varchar2(8);
   v_empno emp.empno%type;
   v_ename  emp.ename%type;
   v_sal   emp.sal%type;
begin
   if updating then
        oper:='Update';
   end if;
   if deleting then
          oper:='Delete';
   end if;

   v_empno:=old.empno;
   v_ename:=old.ename;
   v_sal:=old.sal;
   insert into emp_log values(v_empno,v_ename,v_sal,oper,user,sysdate);
end;
```

Example 2:

- Suppose that instead of allowing negative account balances, the bank deals with overdrafts by
  - setting the account balance to zero
  - creating a loan in the amount of the overdraft providing same loan number as a account number of the overdrawn account

```
create trigger overdraft-trigger
 after update on account
referencing new row as nrow
   for each row
     when nrow.balance < 0
begin
        insert into borrower
                (select customer-name, account-number
                 from depositor
                 where nrow.account_number =  depositor.account-number);
            insert into loan values
                 (nrow.account_number, nrow.branch-name, nrow.balance);
            update account
                    set balance = 0
          where account.account_number = nrow.account_number
end;
```

5.6 Functional Dependencies

- Functional dependencies are constraints on the set of legal relations. It defines attributes of relation, how they are related to each other.
- It determines unique value for a certain set of attributes to the value for another set of attributes that is functional dependency is a generalization of the notation of key.
- Functional dependencies are interrelationship among attributes of a relation.

Definition:

For a given relation R with attribute X and Y, Y is said to be functionally dependent on X, if given value for each X uniquely determines the value of the attribute in Y. X is called determinant of the functional dependency (FD) and functional dependency denoted by X→ Y.

Example 1: consider a relation supplier

Supplier(supplier_id#,sname,status,city)

Here, sname, status and city are functionally dependent on supplier_id. Meaning is that each supplier id uniquely determines the value of attributes supplier name,supplier status and city This can be express by

Supplier.supplier_id→supplier.sname
Supplier.supplier_id→supplier.status
Supplier.supplier_id→supplier.city

Or simply,
supplier_id→ sname
supplier_id→ status
supplier_id→city

Question: is following functional dependency is valid ?
sname→status
sname→city

Answer: it is true only if sname is unique, otherwise false.

Valid case

sname	status
X	Good
Y	Good

Invalid case

sname	status
X	Good
Y	Good
X	Bad

Example 2: Consider a relation student-info

Student-info(name#,course#,phone_no,major,prof,grade)

That is, {name,course} is composite primary key

This relation has the following functional dependencies
{name→phone_no, name→major, name,course→grage, course→prof}

Functional dependency X→Y satisfied on the relation R/ hold on R

FD X→Y is satisfied on relation R if the cardinality of $\prod_Y(\sigma_{x=x}(r))$ is at most one. That is if, two tuples t_i and t_j of R have the same X value then the corresponding value of Y must identical.

Let R be a relational schema
$\alpha \subseteq$R and $\beta \subseteq$R
then the functional dependency $\alpha \rightarrow \beta$ holds on R iff for any legal relation r(R), whenever any two tuples t1 and t2 of r agree on the attributes α then they also agree on the attributes β. That is, if $t_1[\alpha]$=t2[α] then t1[β]=t2[β].

5.6.1 Application of Functional dependencies

Functional dependencies are applicable
- To test the relation whether they are legal under a given set of functional dependency.
  - Let r is a relation and F is a given set of functional dependencies. If r satisfies F, then we determine that r is legal under a given set of functional dependency F
- To specify the constraints for the legal relation
  - We say that f holds on R if all legal relations on R satisfy the set of functional dependencies F.

5.6.2 Types of Functional Dependencies

Trivia functional dependency

Functional dependencies are said to be trivial if it satisfied by all relations.
For example:
- A→A is trivial. It satisfied by all relation involving attribute A
- AB→A is trivial. It satisfied by all relations involving attribute A.

In general, A functional dependency of the form $\alpha \rightarrow \beta$ is trivial if $\beta \subseteq \alpha$.
Verification:

Consider a relation r

	A	B	C	D
	a_1	b_1	c_1	d_1
t_1 →	a_1	b_2	c_1	d_2
	a_2	b_2	c_2	d_2
t_2 →	a_2	b_2	c_2	d_3
	a_3	b_3	c_2	d_4

$t_1[AB]=a_2b_2$, $t_2[AB]=a_2b_2$ agree

$t_1[A]=a_2$ $t_2[A]=a_2$ agree

Here, $t_1[AB]=t_2[AB] \Rightarrow t_1[A]= t_2[A]$. This implies AB→A is satisfied.

Fully functionally dependency

For a given relation schema R, FD X→Y, Y is said to be fully functionally dependent on X if there is no Z (where Z is a proper subset of X) such that Z→Y.

Example: Let us consider relational schema R=(A,B,C,D,E,H) with the FDs
F={A→BC,CD→E,C→E,CD→E,CD→AH, ABH→BD,DH→BC}

- Here, the FD A→BC is left reduced, so clearly, BC is fully functionally dependent on A (because there is no possible proper subset of only element A)
- Here, the FDs CD→E, C→E where E is functionally dependent on CD and again E is functionally dependent on subset of CD. That is C (i.e. C→E). Hence E is not fully functionally dependent on CD.

Example: Consider a relation sales

Sales (product_id#,sales_date#,quantity,product_name)
With the following functional dependencies

F={product_id,sales_date→quantity, product_id→quantity, product_id→product_name}

- Here,. FDs product_id,sales_date→quantity, product_id→quantity, quantity is not fully functional dependent on product_is,sales_date.
- Here, functional dependency product_id→product_name, product_name is fully functional dependent on product_id.

Partial functional dependency

For a given relation schema R with set of functional dependency F on attribute of R. Let K as a candidiate key in R. if X is a proper subset of K and X and X→A then A is said to be partially dependent on K.

Example: Consider a relation schema 'student_course_info'

student_course_info(name#,course#,grade,phone_no,major,course_department)

with the following FDs

{name→phone_no,major
course→course_department,
name,course→grade
}

Here {name,course} is a candidate key. Here grade is fully functionally dependent on {name,course}. If thee is a possible FD name→grade then we can not say grade is fully

functionally dependent on {name,course}. Here phone_no, major and course_department are partially dependent on {name,course}

Transitive dependency

For a given relational schema R with set of functional dependency F. Let X and Y be the subset of r anf Let A be the attribute of R s.t. $X \not\subset Y$, $A \not\subset XY$. If the functional dependencies {X→Y, Y→A} implies by F (i.e. X→Y→A) then A is said to be transitively dependent on X.

Example:
Let us consider relational schema 'prof_info'
prof_info=(prof_name#,department_name, head_of_department)

with the set functional dependency
F={prof_name→department_name, department_name→head_of_department}

Here prof_name→department_name→head_of_department so head_of_department is transitively dependent on the key prof_name.

Example:
Let R=(A,B,C,D,E) and FDs F={AB→C,B→D,C→E}
Here AB act a candidate key and E is transitively dependent on the key AB, ince AB→C→E).

5.6.3 Closure of Set of Functional Dependencies

For a given set of functional dependencies F, there are certain other functional dependencies that are logically implies by F. (i.e. if A→B and B→C, then we can write A→C). the set of all functional dependencies logically implies F is the closure of F. Closure of F is denoted by F^+.

We can find all of F^+ by applying Armstrong's Axioms:
- if $\beta \subseteq \alpha$ then $\alpha \rightarrow \beta$ or $\alpha \rightarrow \alpha$ (reflexive)
- if $\alpha \rightarrow \beta$ then $\gamma\ \alpha \rightarrow \gamma\ \beta$ (augmentation)
- if $\alpha \rightarrow \beta$ and $\beta \rightarrow \gamma$ then $\alpha \rightarrow \gamma$ (transitivity)

Example: Let R=(A,B,C,G,H,I)
F={A→B, A→C,CG→H,CG→I,B→H}
Compute closure of F^+.

Closure of F^+ computed as follow:
- A→H
  - by transitivity A→B and B→H
- AG→I
  - By augmenting A→C with G we get AG→CG and then by transitivity with CG→I we get AG→I
- CG→HI
- From CG →H and CG→I "union rule" can be inferred from definition of functional dependency ot
  Augmentation of CG→I to infer CG→CGI, argumentation of CG→H to infer CGI→HI, and then transitivity.

Hence, F^+={ A→A,B→B,C→C,H→H,G→G,I→I,A→B,

A→C,CG→H,CG→I,CG→HI,B→H,A→H,
AG→I,CG→Hi
}

here , first six FDs obtain by reflexive axiom.

We can further simplify the the computation of F+ by using the following addition rule.

(a) if $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds, then $\alpha \rightarrow \beta\gamma$ (Additivity or union rule)
(b) if $\alpha \rightarrow \beta\gamma$ holds then $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds (projectivity/decomposion)
(c) if $\alpha \rightarrow \beta$ holds and $\gamma\beta \rightarrow \delta$ holds then $\alpha\gamma \rightarrow \delta$ holds (pseudotransitivity)

Examples: Let R=(A,B,C,D) and F={A→B,A→C,BC→D} then compute F^+.

- Since A→B and A→C then by union rule A→BC.
- Since BC →D, then by projective/decomposition B→D, C→D. Again by transitivity A→B & B→D $\Rightarrow$ A→D and A→C and C→D $\Rightarrow$ A→D.
- Hence, F+ ={A→A, B→B, C→C, D→D, A→B, A→C, BC→D, B→D, C→D, A→D}

5.6.4 Attribute Closure

The closure of X under a set of functional dependencies F, written as X+, is the set of attributes {A1,A2, . . Am} such that the FD X→Ai for Ai$\in X^+$ follows from F by the inference axioms for functional dependencies.

Example:

Let X=BCD and F={A→BC,CD→E,E→C,D→AEH,ABH→BD,DH→BC}. Compute the closure X^+ of X under F.

- initialize X^+:=BCD.
- Since left hand side of the FD CD→E is a subset of X+ (i.e CD$\subseteq X^+$), X^+ is augmented by the right hand side of the FD (i.e. E) thus now X+:=BCDE.
- Similarly, D$\subseteq X^+$, the right hand side of the FD D→AEH is added to X^+. Hence now X+:=ABCDEH.
- Now X^+ can not be augmented any further because no FDs left hand side is subset of X^+.

Application of Attribute Closure

1. Testing superkey
   To test α is a superkey we compute α^+ and check whether α^+ contains all attributes of R. if so α is a superkey, otherwise not.

2. Testing functional dependencies
   To check a functional dependency $\alpha \rightarrow \beta$ holds check whether $\beta \subseteq \alpha^+$. If so $\alpha \rightarrow \beta$; otherwise not.