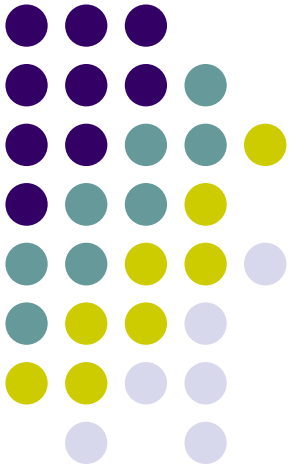
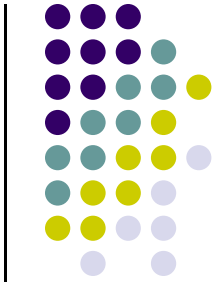
Chp 2: Integrity and Security

Slides by: Shree J.



Topics to be covered

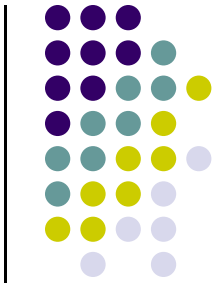


- Domain constraints
- Referential integrity
- Assertions
- Triggers
- Assertions and triggers in SQL
- Security and authorization
- Authorization in SQL

Integrity Constraints

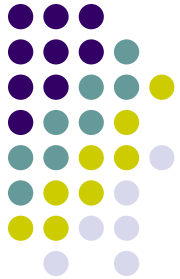
- Integrity constraints guard against accidental damage to the database, by ensuring that authorized changes to the database do not result in a loss of data consistency.
  - A checking account must have a balance greater than Rs.10,000.00
  - A salary of a bank employee must be at least Rs.200.00 an hour
  - A customer must have a (non-null) phone number

Integrity Constraints



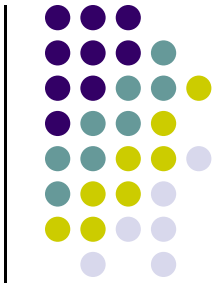
- It is a mechanism used to prevent invalid data entry into the table.
- Used for enforcing rules that the columns in a table have to confirm with
- Types of integrity constraints
  - Domain integrity constraints
  - Entity integrity constraints
  - Referential integrity constraints

Domain Integrity constraints



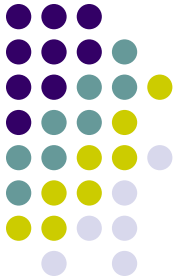
- Constraints set a range, and any violations that take place will prevent the user from performing the manipulation.
  - Not Null constraint
  - Check constraint

Not Null constraint



- By default the table can contain null values.
- The enforcement of Not Null in a table ensures that the table contains values.
- Not Null can be defined using alter table command even when the table contains rows.
- The table can be altered only if the column being modified contains not null values.
- Note:- Zero and Null are not equivalent.

Check constraints



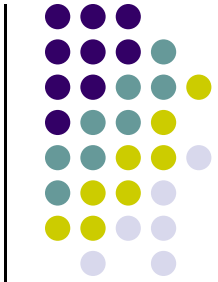
- Specify conditions that each row must satisfy
- Rules are governed by logical expressions or Boolean expressions
- Cannot contain subqueries.

  Create table abc(a number(2) constraint aa check(a>10), b varchar2(15),

  c date);---during table creation

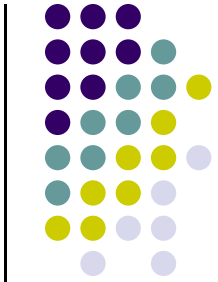
  Alter table abc add constraint aa check (a>10);---after table creation

Table Level constraint



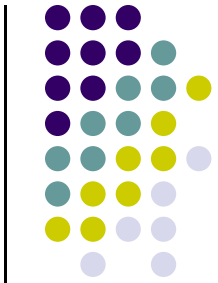
- IC defined at table level can impose rules on any columns in the table.
- Not null can be given only at the column level

Entity Integrity constraints

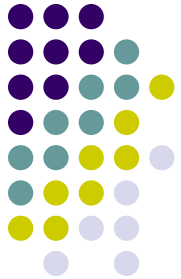


- Each row in a table can be uniquely identified using the entity constraint
  - Unique constraints
  - Primary key constraints

Unique constraints



- Used to prevent the duplication of values within the rows of specified column or a set of columns in a table.
- This constraint can also allow Null values.
- If unique key is defined in more than one column then it is said to be composite unique key.
- Can be applied only at table level.

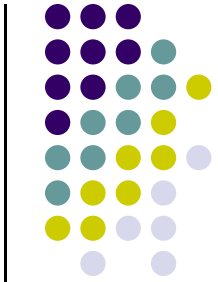


- Alter table abc add constraint dd unique(c);
- Create table abc(a number(2) not null,
  b varchar2(15) unique, c date);

Primary Key constraints

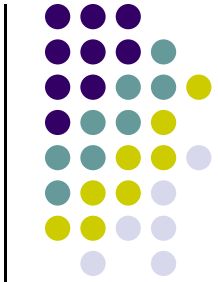
- Avoids duplication of rows and does not allow null values, when enforced in a column or set of columns.
- Used for identification of a row.
- A table can have only one primary key
- Can be created during table creation or using alter table
- Note:- **cannot be defined in an alter table command when the table contains rows having Null values.**
- Create table abc(a number(2), b varchar2(15), c date, constraint a_prime primary key(a));

Referential Integrity constraints



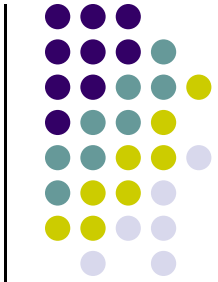
- To establish a ‘parent-child’ or a ‘master-detail’ relationship between two tables having a common column, a referential integrity constraint is used.
- This can be implemented the column in the parent table as a primary key and the same column in the child table as a foreign key referring to the corresponding parent entry.

Basic concepts related to referential integrity



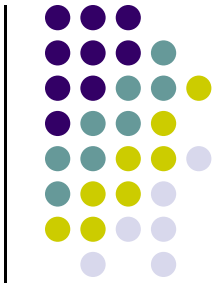
- Foreign key:- Column(s) included in the ref. integrity refer to a referenced key
- Referenced key:- It is a unique or a primary key defined on the column belonging to the parent table.
- Child table:- depends upon the values present in the referenced key of the parent table.
- Parent table:- Determines whether insertion or updation of data can be done in child table

At the time of table creation



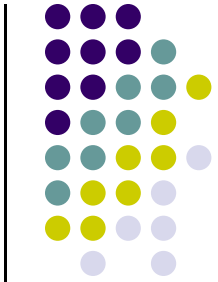
- Create table dept(deptno number(2) primary key, dname varchar2(15) unique, loc varchar2(15) not null);
- Create table emp(empno number(2) primary key, ename varchar2(15) not null, salary number(7,2) not null, deptno number(2) constraint fk_Dept references dept(Deptno));

Constraints on a Single Relation



- **not null**
- **primary key**
- **unique**
- **check** (*P*), where *P* is a predicate

Not Null Constraint



- Declare *branch_name* for *branch* is **not null**

  branch_name **char**(15) **not null**

- Declare the domain *Dollars* to be **not null**

  create domain *Dollars* **numeric**(12,2) **not null**

The Unique Constraint

- **unique** (A_1, A_2, …, A_m)
- The unique specification states that the attributes

  $A1$, $A2$, … Am

  form a candidate key.
- Candidate keys are permitted to be null (in contrast to primary keys).

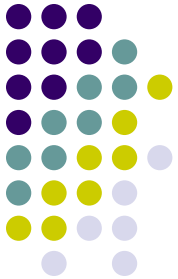
The check clause

- **check** (*P*), where *P* is a predicate

Example: Declare *branch_name* as the primary key for *branch* and ensure that the values of *assets* are non-negative.

```
create table branch
        (branch_name     char(15),
         branch_city     char(30),
         assets          integer,
         primary key (branch_name),
         check (assets >= 0))
```

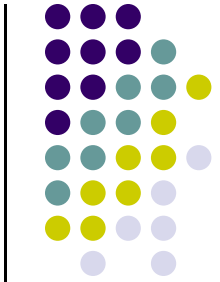
The check clause (Cont.)



- The **check** clause in SQL-92 permits domains to be restricted:
  - Use **check** clause to ensure that an hourly_wage domain allows only values greater than a specified value.

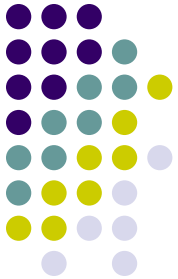
    create domain *hourly_wage* **numeric(5,2)**
    constraint *value_test* **check**(*value* > = 200.00)
  - The domain has a constraint that ensures that the hourly_wage is greater than 200.00
  - The clause **constraint** *value_test* is optional; useful to indicate which constraint an update violated.

Referential Integrity



- 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 “Perryridge” is a branch name appearing in one of the tuples in the *account* relation, then there exists a tuple in the *branch* relation for branch “Perryridge”.
- Primary and candidate keys and foreign keys can be specified as part of the SQL **create table** statement:
  - The primary key clause lists attributes that comprise the primary key.
  - The unique key clause lists attributes that comprise a candidate key.
  - The foreign key clause lists the attributes that comprise the foreign key and the name of the relation referenced by the foreign key. By default, a foreign key references the primary key attributes of the referenced table.

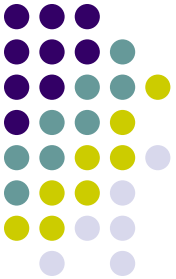
Referential Integrity in SQL – Example



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 **numeric**(12,2),
primary key (*branch_name*))

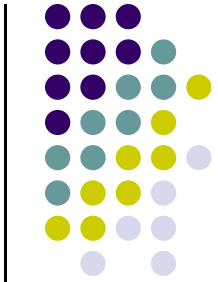
Referential Integrity in SQL – Example (Cont.)



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*)

Assertions



- An **assertion** is a predicate expressing a condition that we wish the database always to satisfy.
- An assertion in SQL takes the form

  create assertion <assertion-name> **check** <predicate>
- When an assertion is made, the system tests it for validity, and tests it again on every update that may violate the assertion
  - This testing may introduce a significant amount of overhead; hence assertions should be used with great care.
- Asserting

  for all X, $P(X)$

  is achieved in a round-about fashion using

  not exists X such that not $P(X)$

Assertion Example

- Every loan has at least one borrower who maintains an account with a minimum balance or Rs.1000.00

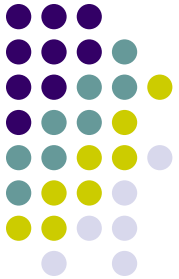
```
create assertion balance_constraint check
    (not exists (select *
            from loan
           where not exists (select *
              from borrower, depositor, account
              where loan.loan_number = borrower.loan_number
                 and borrower.customer_name =
depositor.customer_name
                 and depositor.account_number =
account.account_number
                 and account.balance >= 1000)))
```

Assertion Example

- The sum of all loan amounts for each branch must be less than the sum of all account balances at the branch.

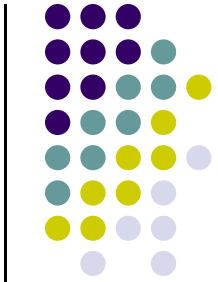
```
create assertion sum_constraint check
    (not exists (select *
                 from branch
                 where (select sum(amount )
                        from loan
                        where loan.branch_name =
                              branch.branch_name )
                       >=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

SQL Triggers



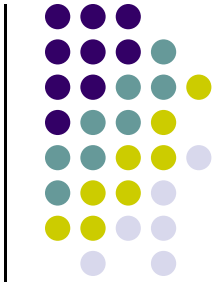
- Triggers are stored programs that may be configured to automatically execute or fire when certain event takes place.
- A user can write triggers on a table or a view.
- They are used to implement complex validation, security and auditing, and replication of data
- To enforce referential integrity and general integrity constraints

SQL Triggers



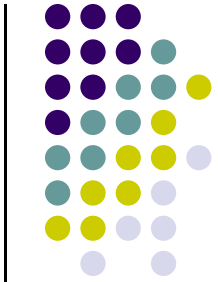
- Objective: to monitor a database and take action when a condition occurs
- Triggers are expressed in a syntax similar to assertions and include the following:
  - event (e.g., an update operation)
  - condition
  - action (to be taken when the condition is satisfied)
- Generally, the trigger name should indicate the table name, operation & type of trigger i.e. row trigger or statement trigger.

Types of Triggers



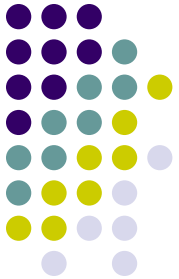
- A **statement-level trigger** is assumed if you omit the FOR EACH ROW keywords.
- This type of trigger is executed once, before or after the triggering statement is completed.
- This is the default case.

Types of Triggers



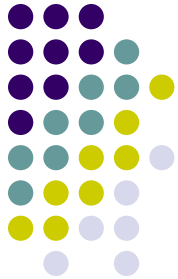
- A **row-level trigger** requires use of the FOR EACH ROW keywords.
- This type of trigger is executed once for each row affected by the triggering statement.
- In other words, if you update 10 rows, the trigger executes 10 times.

Syntax of trigger



```
create or replace trigger <TRIGGER_NAME>
   before insert or update
on <table_name>
   for each row
declare <VARIABLE DECLARATIONS>
   begin <CODE>
exception <EXCEPTION HANDLERS>
   end <TRIGGER_NAME>;
```

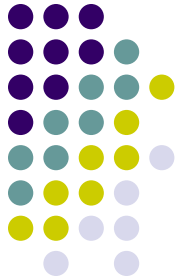
SQL Triggers: An Example



- A trigger to compare an employee's salary to his/her supervisor during insert or update operations:

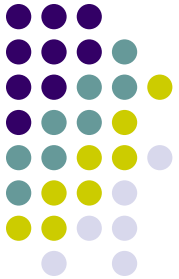
```
CREATE TRIGGER INFORM_SUPERVISOR
    BEFORE INSERT OR UPDATE OF
    SALARY, SUPERVISOR_SSN
ON EMPLOYEE
    FOR EACH ROW
WHEN(NEW.SALARY> (SELECT SALARY FROM EMPLOYEE
    WHERE SSN=NEW.SUPERVISOR_SSN))
INFORM_SUPERVISOR NEW.SUPERVISOR_SSN,NEW.SSN;
```

Referring *:new* and *:old* values



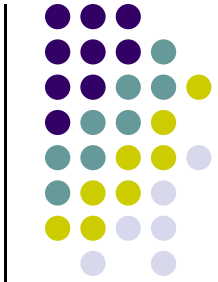
- For INSERT, :new can be used because values are new
- For DELETE, :old can be used because values are from an existing row
- For UPDATE, both :new and :old can be used because the row will have both old and new values

Another example



```
CREATE OR REPLACE TRIGGER author_trig
AFTER UPDATE OF first_name
ON authors
FOR EACH ROW
WHEN (OLD.first_name != NEW.first_name)
BEGIN
DBMS_OUTPUT.PUT_LINE('First Name '
||:OLD.first_name
||' has change to '
||:NEW.first_name);
END;
/
```

Another example (contd.)



- **To test the trigger, simply update the first name of one of the authors.**

  SET SERVEROUTPUT ON

  UPDATE authors

  SET first_name = 'Ronald'

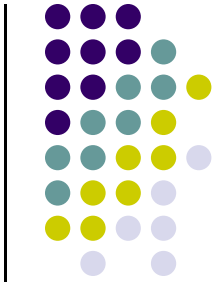
  WHERE first_name = 'Ron';

- **The trigger immediately fires and displays the following on the screen:**

  First Name Ron has change to Ronald

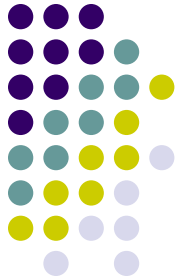
- **The trigger fired as expected, and the PL/SQL was executed.**

Restriction on trigger code



- You cannot write any transaction control statement, like *Commit*, *Rollback* and *Savepoint*;
- Inside the trigger code,no DDL statement can be executed directly or indirectly via procedures in order to support statement level atomicity.
- If the row trigger is not properly written then it generates the effect which depends on the order in which rows are processed.

Altering trigger



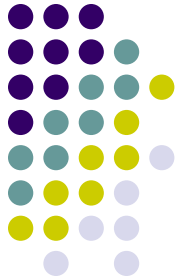
- A trigger once created cannot be altered
- If change wanted then it can be done by dropping the trigger & then recreating it.
- It can also be done by creating trigger using CREATE or REPLACE command as follows:

DROP TRIGGER t1

or

CREATE OR REPLACE TRIGGER t1

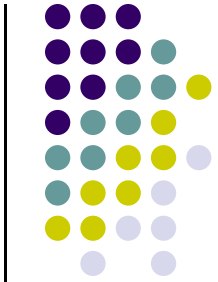
Enabling and Disabling trigger



- A trigger can be disabled temporarily when:
  - Referenced objects are not available
  - Large data is loaded in the table
- Trigger is disabled/enabled using the ALTER TRIGGER command
  - ALTER TRIGGER TRG1 DISABLE;
  - ALTER TABLE T1
    DISABLE ALL TRIGGERS

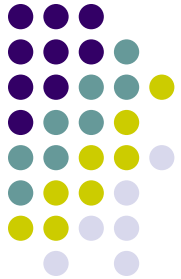
  - ALTER TRIGGER TRG1 ENABLE
  - ALTER TABLE T1
    ENABLE ALL TRIGGERS

Information on triggers



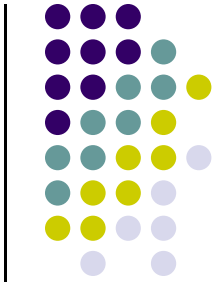
- Information about all triggers is available from views
  - USER_TRIGGERS
  - ALL_TRIGGERS
  - DBA_TRIGGERS

Why Security?



- The data stored in the database need protection from unauthorized access and malicious destruction or alternation.
- Protection against accidental introduction of inconsistency that integrity constraints provide.

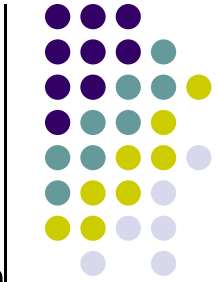
Security Violations



- Forms of malicious access are
  - Unauthorized reading of data
  - Unauthorized modification of data
  - Unauthorized destruction of data.

- Database Security refers to protection from malicious access.
  - ➢Security measures at the database system level.
  - ➢Security measures at the OS level
  - ➢Security measures at the Network level
  - ➢Security measures at the Physical level
  - ➢Security measures at the Human level

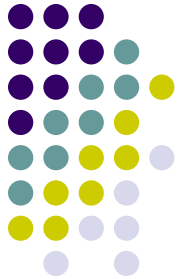
Authorization



- Several forms of authorization can be assigned to a user.
  - Read authorization
  - Insert authorization
  - Update authorization
  - Delete authorization
- Authorization for the modification of database schema
  - Index authorization
  - Resource authorization
  - Alteration authorization
  - Drop authorization

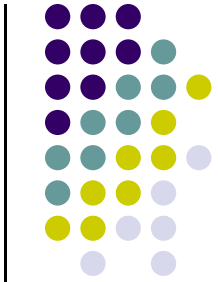
Database admin has the ultimate authority to authorize new users/restructure the database

Granting of privileges



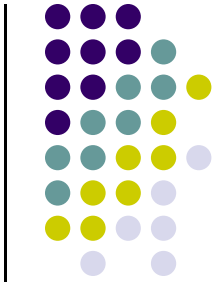
- Authorization can be granted using grant command.
- The passing of authorization from one user to another is represented by authorization graph.
- In order to maintain security it is required that all edges in an authorization graph be part of some path originating with the database administrator.

Roles



- Authorizations given to specific users are called as roles.
- A set of roles is created in the databases.
- Authorization can be granted to roles just like ones granted to individual users.

Audit trail



- It is a log of all changes (inserts/deletes/updates) to the database, along with information such as which user performed the change and when the change was performed.
- Can be created by triggers

Thank You!