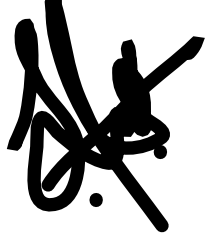
Chapter 5
Integrity Constraints

Integrity Constraints

- Domain Constraints
- Referential Integrity
- Triggers
- Assertion



Domain Constraints (1)

- 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.
- Domain constraints are the most elementary form of integrity constraint.
- They test values inserted in the database, and test queries to ensure that the comparisons make sense.
- New domains can be created from existing data types

  E.g. create domain Dollars numeric(12, 2)

  create domain Pounds numeric(12,2)

Domain Constraints (2)

- The check clause in SQL 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 > = 4.00)
```

- The domain has a constraint that ensures that the hourly-wage is greater than 4.00
- The clause constraint value-test is optional; useful to indicate which constraint an update violated.

- The check clause can be used to restrict a domain to not contain any null values.

```
create domain  AccountNumber char(10)
        constraint account_number_null_test
                check(value not null)
```

The domain can be restricted to contain an specified set of values using in clause.

```
create domain  AccountType char(10)
        constraint account_type_test
                check(value in ('Saving', 'Current') )
```

The check condition above, can be more complex, since subqueries that refers to other relations are permitted.

```
check(value in (select branch_name from branch) )
```

Referential Integrity (1)

- 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".

Referential Integrity (2)

- Referential integrity is a database concept that ensures that relationships between tables remain consistent.
- When one table has a foreign key to another table, the concept of referential integrity states that you may not add a record to the table that contains the foreign key unless there is a corresponding record in the linked table.
- It also includes the techniques known as cascading update and cascading delete, which ensure that changes made to the linked table are reflected in the primary table.

Employees and Managers tables

- Consider the situation where we have two tables:
- The Employees table has a foreign key attribute entitled ManagedBy which points to the record for that employee's manager in the Managers table. Referential integrity enforces the following three rules:
- We may not add a record to the Employees table unless the ManagedBy attribute points to a valid record in the Managers table.
- If the primary key for a record in the Managers table changes, all corresponding records in the Employees table must be modified using a cascading update.
- If a record in the Managers table is deleted, all corresponding records in the Employees table must be deleted using a cascading delete.

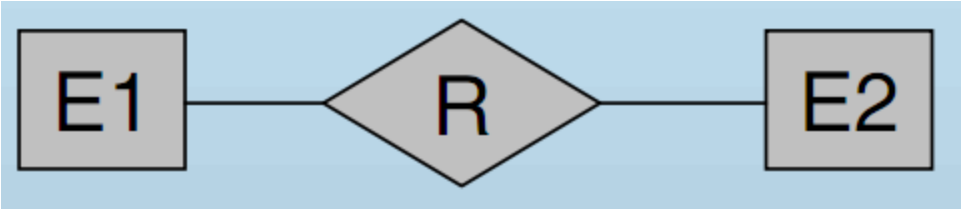
Formal Definition

- 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 r_1 such that $t_1[K_1] = t_2[\alpha]$.
- Referential integrity constraint is also called subset dependency since its can be written as

$$\Pi_{\alpha}(r2) \subseteq \Pi_{K_1}(r1)$$

Referential Integrity and the 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



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

Referential Integrity in SQL (1)

- Primary and candidate keys and foreign keys can be specified as part of the SQL create table statement.
- The primary key clause of the create table statement includes a list of the attributes that comprise the primary key.
- The unique key clause of the create table statement includes a list of the attributes that comprise a candidate key.

Referential Integrity in SQL (2)

- By default, a foreign key references the primary key attributes of the referenced table.
- The foreign key clause of the create table statement includes
  - a list of the attributes that comprise the foreign key
  - and the name of the relation referenced by the foreign key using references clause.

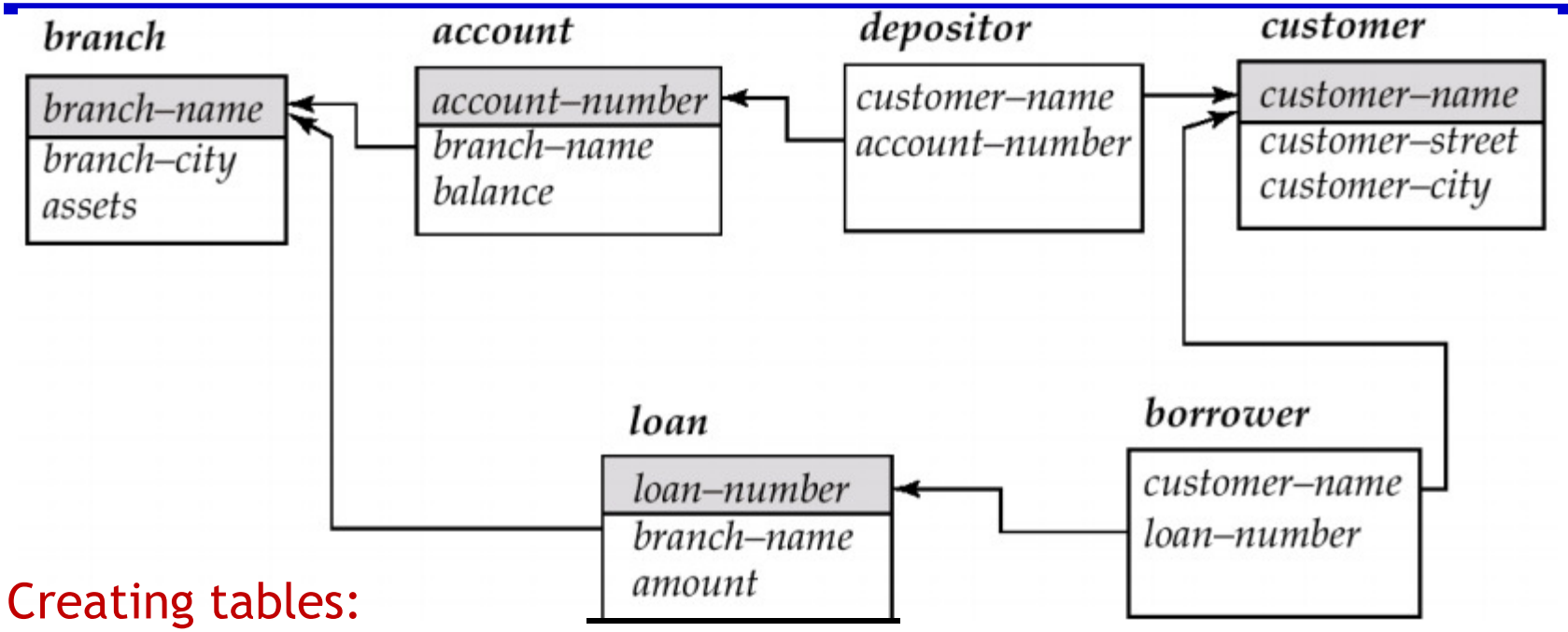
Referential Integrity in SQL (3)

- When a referential integrity constraint is violated, the normal procedure is to reject that action that caused the violation.

- However, a foreign key clause can specify that if a delete or update action on the refereneced relation violates the constraint then

```
create table account
(...
        foreign key(branch_name) references branch
                                        on delete cascade
                                        on update casade,
.....
)
```

Schema Diagram for the banking enterprise



Creating tables:

```
create table account
(account_number char(10),
Branch_name char(15),
balance integer,
primary key (account_number),
foreign key (branch_name) references branch)
```

Referential Integrity in SQL (4)

- Due to the on delete cascade clauses, if a delete of a tuple in branch results in referential-integrity constraint violation, the delete "cascades" to the account relation, deleting the tuple that refers to the branch that was deleted.
- Similarly the system does not reject an update to a field referenced by the constraint if it violates the constraint, instead the system updates the field branch_name in the referenced tuple in account to the new value as well.
- Also possible to specify to set null or set default in place of cascade.
- Attributes of foreign keys are allowed to be null if they have not been declared to be non-null.

Assertion

- An assertion is a predicate expressing a condition that we wish the database always to satisfy.
- Domain constraints and referential-integrity constraints are special form of assertions.
- 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**.

Assertion examples

- Total number of students in each program must be less or equal to total number of students in college.

```
create assertion check_program check
        (select sum(no_of_students) from program
                <=
         select count(*) from college)
```

- Whenever some tuples in table/database cause the condition of assertion statement to be evaluated to false, the constraint is violated.

Trigger

- A trigger is a statement that the system executes automatically as a side effect of a modification of the database.
- To design a trigger mechanism, we must consider event-condition-action model:
  - Specify the conditions under which the trigger is to be executed.
  - Specify the actions to be taken when the trigger executes.

Assertions - An assertion is a piece of SQL which makes sure a condition is satisfied or it stops action being taken on a **database object**.

Triggers - a trigger is a piece of SQL to execute either before or after an update, insert, or delete in a database.

Trigger example

- 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
  - ❖ giving this loan a loan number identical to the account number of the overdrawn account
- The condition for executing the trigger is an update to the *account* relation that results in a negative balance value

Trigger basic syntax (1)

```
CREATE TRIGGER <triggerName>
BEFORE | AFTER <triggerEvent> ON <TableName>
[REFERECING <oldOrNewValuesAliasList>]
[FOR EACH {ROW | STATEMENT}]
[WHEN (<triggerCondition>)]
<triggerBody>
```

Tigger Events:
Trigger events can be insert, delete or update.

Types of Triggers:

A row-level trigger fires once for each row that is affected by a triggering event. For example, if deletion is defined as a triggering event on a table and a single DELETE command is issued that deletes five rows from the table, then the trigger will fire five times, once for each row.

A statement-level trigger fires once per triggering statement regardless of the number of rows affected by the triggering event. In the prior example of a single DELETE command deleting five rows, a statement-level trigger would fire only once.

Trigger basic syntax (2)

Before | After: Triggers can be activated before or after the event insert/ delete/update.

[REFERECING <oldOrNewValuesAliasList>]

- The referencing old row as clause can be used to create a variable storing the old value of an updated or deleted row.
- The referencing new row as clause can be used with inserts and updates.
- The clause referencing old table as or referencing new table as can be used to refer to temporary tables (called tranistion tables) containing all the affected rows.
  - Transition table cannot be used with before trigger.
  - But it can be used with after trigger, regardless of whether they are statement trigger or row trigger.

- Time: {**BEFORE** / **AFTER**} {**INSERT** / **DELETE** / **UPDATE**}
- Execution:
  - FOR EACH ROW: row-level trigger
  - FOR EACH STATEMENT (default): only once for the entire event
- <oldOrNewValuesAliasList>:
  - OLD/NEW or OLD ROW/NEW ROW: row-level trigger
  - OLD TABLE/NEW TABLE: AFTER trigger
  - no old values for INSERT events, no new values for DELETE events
- WHEN: statement specifies the condition which is optional.

Trigger example cont..

```
create trigger overdraft_trigger after update on account
referencing new row as nrow
for each row
when nrow.balance < 0
begin atomic
      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
```

Trigger event and action in SQL

- Triggering event can be insert, delete or update
- Triggers on update can be restricted to specific attributes

  E.g. create trigger overdraft_trigger after update of balance on account
- Values of attributes before and after an update can be referenced
  - referencing old row as : for deletes and updates
  - referencing new row as : for inserts and update
- Triggers can be activated before an event, which can serve as extra constraints.
  - E.g. convert blanks to null.