Database Administration:
The Complete Guide to Practices and Procedures

Chapter 8

Data Integrity

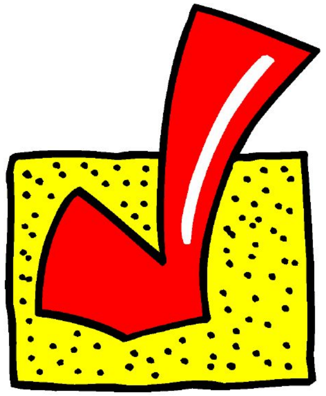
Agenda

- Types of Integrity
- Database Structure Integrity
- Semantic Data Integrity
- Questions

Types of Integrity

With respect to databases, we will discuss two aspects of integrity:

- Database structure integrity
  - Keeping track of database objects and ensuring that each object is created, formatted, and maintained properly
- Semantic data integrity
  - Assuring the correct meaning of data and the relationships that need to be maintained between different types of data



http://craigsmullins.com/dbta_039.htm

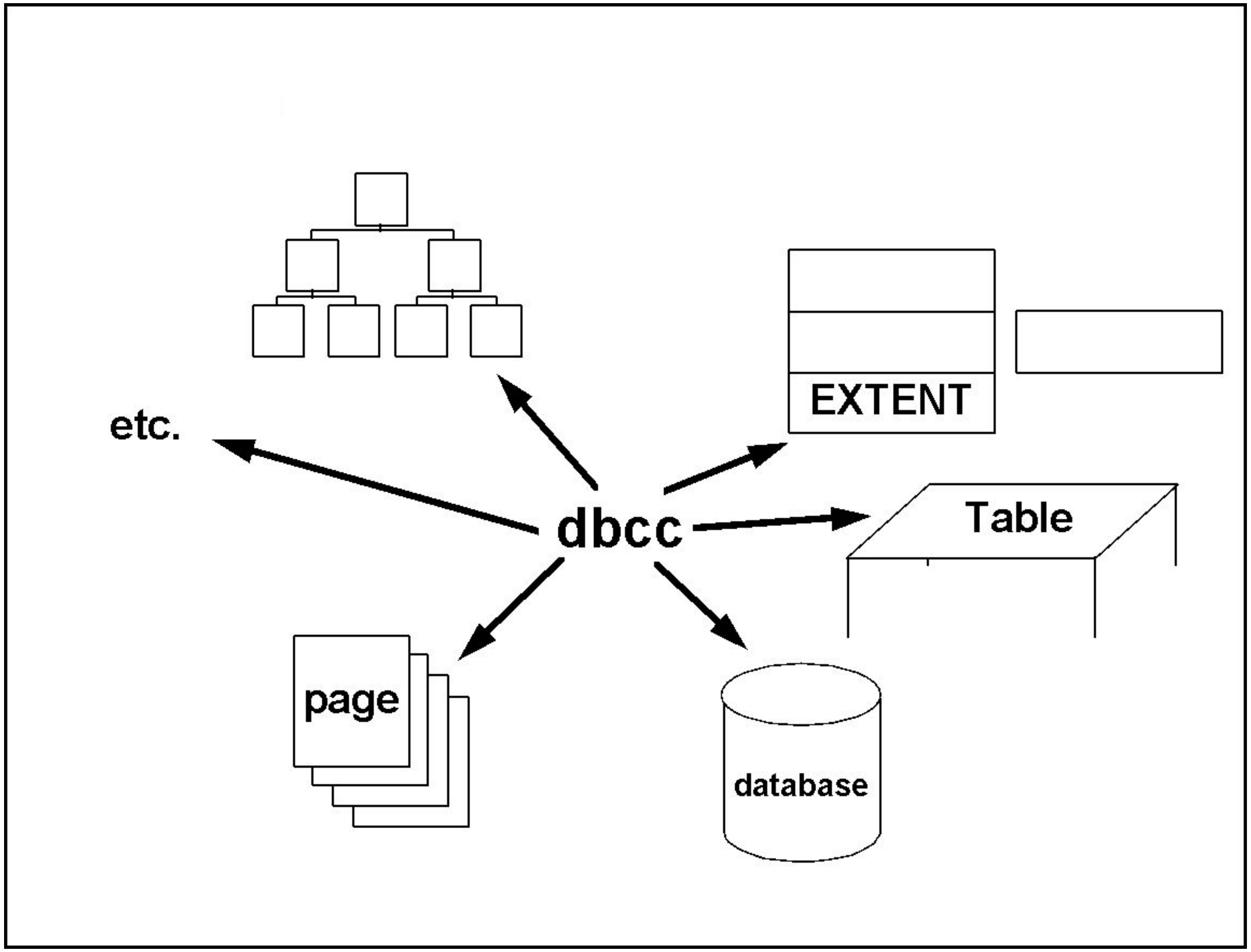
Database Structure Integrity

- Index corruption
  - If the pointers do not point to the correct data, the index is useless.
- Other types of pointer corruption
  - XML, LOB, large text data, etc.
- Page header corruption
  - If the header becomes corrupted, the DBMS may not be able to interpret the data stored on the page.
- Backup files
  - If the backup file is not formatted correctly, or if data is in the wrong location in the backup file, it cannot be used by the DBMS for recovery purposes.
  - Media failures, tape degradation, and bugs can cause such problems.

Managing Structural Problems

- You can investigate the integrity of a database using DBMS utility programs.
- Examples include:
  - dbcc (Microsoft SQL Server & SYBASE)
  - CHECK and REPAIR (DB2)
  - TBCHECK (Informix)
- We will examine dbcc functionality as an example of the type of things that these utilities can accomplish.

Using dbcc to Check for Structural Problems



dbcc: Consistency Options

- DBCC CHECKTABLE(table_name)
  - checks the consistency of the data and index pages of a table. When DBCC is run using this option it will report on the number of data pages, the number of rows, the number of text and image columns, as well as any integrity violations.
- DBCC REINDEX(table_name)
  - defragments the indexes built on the specified table.

dbcc: Database Checking

- DBCC CHECKDB(database_name)
  - Runs CHECKTABLE on every table in the database.
- DBCC CHECKCATALOG(database_name)
  - Checks the consistency of the system catalog tables; reports on the size and number of segments used, and detect any integrity errors.
- DBCC CHECKALLOC(database_name)
  - Checks the consistency of the specified database and reports on the current extent structure. This option also reports on the number of allocations and the pages used per allocation.
- DBCC CHECKFILEGROUP(filegroup_name)
  - Checks the allocation and structural integrity of all tables and indexed views in the specified database and issues a report of the findings.

dbcc: Memory Usage

- DBCC MEMUSAGE will report on the configured memory allocation and memory usage of the top 20 memory users.
- It shows:
  - Configured memory
  - Code size
  - Kernel and structures
  - Page cache
  - Procedure buffers and headers
  - Buffer cache detail
  - Procedure cache detail

dbcc: Other Options

The DBCC utility can also be used:

- To generate reports containing information on database internals (for example, creation date and internal identifier)
- To print formatted table pages showing the header contents, to dump and display the contents of buffers and caches
- To “zap” the database (that is, make quick changes to any of the database contents).

Semantic Data Integrity

Many forms of semantic data integrity can be enforced by using features of the DBMS.

- Entity Integrity
- Uniqueness
- Data Types
- Default Values
- Check Constraints
- Triggers
- Referential Integrity

Entity Integrity

- *Entity integrity* means that each occurrence of an entity must be uniquely identifiable.
- Although most DBMSs do not FORCE the creation of a primary key for each table, it is a tenet of the Relational Model.
- Enforce entity integrity by creating a PK for each table in the database.
  - A primary key constraint can consist of one or more columns from the same table that are unique within the table.
  - A table can have only one primary key constraint, which cannot contain nulls.

Uniqueness

- A *unique constraint* is similar to a primary key constraint.
  - However, each table can have *many* unique constraints.
  - Unique constraints cannot be used to support referential constraints.
- The values stored in the column, or combination of columns, must be unique within the table.
  - That is, no other row can contain the same value.
- A unique constraint most likely requires a unique index to enforce.

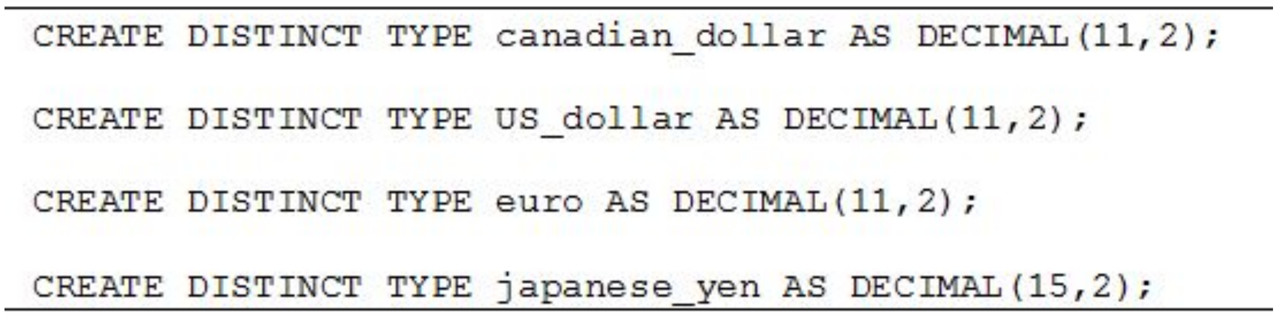
Data Types

- *Data type* and *data length* are the most fundamental integrity constraints applied to data in the database.
  – DBAs must choose data types wisely.
  – The DBMS will automatically ensure that only the correct type of data is stored in that column.
- Choose the data type that most closely matches the domain of values for the column.
  – For example, a numeric column should be defined as one of the numeric data types: integer, decimal, or floating point.
  – If you specify a character data type for a column that will contain numeric data the DBMS cannot automatically enforce the integrity of the data.

http://craigsmullins.com/q-pipe-dq.htm

User-Defined Data Types

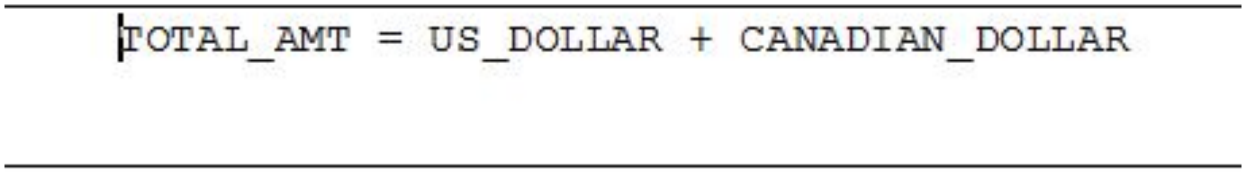
- A *user-defined data type* (UDT) extend the type of data that can be stored in databases and the way that the data is treated.
- UDTs are useful when you need to store data that is specifically tailored to your organization's requirements.
- For example, if your company handles monetary amounts from Canada, the United States, the European Union, and Japan. The DBA can create four UDTs:



```
CREATE DISTINCT TYPE canadian_dollar AS DECIMAL(11,2);

CREATE DISTINCT TYPE US_dollar AS DECIMAL(11,2);

CREATE DISTINCT TYPE euro AS DECIMAL(11,2);

CREATE DISTINCT TYPE japanese_yen AS DECIMAL(15,2);
```

Using UDTs

- After a user-defined data type has been created, it can be used in the same manner as a system-defined data type.
- Strong typing prohibits operations between different data types.
  - For example, the following would NOT be allowed:



```
TOTAL_AMT = US_DOLLAR + CANADIAN_DOLLAR
```

Default Values

- Each column can be assigned a default value that will be used if subsequent INSERTs do not provide a value.
  - Each column can have only one default value.
  - The column's data type, length, and property must be able to support the default value specified.
  - The default may be null, but only if the column is created as a nullable column.

Check Constraints

- A *check constraint* is a DBMS-defined restriction placed on the data values that can be stored in a column or columns.
  - The expression is explicitly defined in the table DDL and is formulated in much the same way that SQL WHERE clauses are formulated.
  - Any attempt to modify the column data (INSERT or UPDATE) causes the expression to be evaluated.
  - If the modification conforms to the expression, the modification is permitted to proceed.
  - If not, the statement will fail with a constraint violation.

http://craigsmullins.com/adi-db2-2.htm

Defining Check Constraints

- The check constraint syntax consists of two components: a constraint name and a check condition.
  - The *constraint name* identifies the check constraint to the database. If a constraint name is not explicitly coded, the DBMS automatically generates a unique name for the constraint.
  - The *check condition* defines the actual constraint logic. The check condition can be defined using any of the basic predicates (>, <, =, <>, <=, >=), as well as BETWEEN, IN, LIKE, and NULL.
    - Furthermore, AND and OR can be used to string conditions together in a check constraint.

Check Constraint Example

```
CREATE TABLE EMP
(empno        INTEGER       PRIMARY KEY,
    CONSTRAINT check_empno
      CHECK (empno BETWEEN 100 and 25000),
 emp_address  VARCHAR(70),
 emp_type     CHAR(8)
      CHECK (emp_type IN ('temp', 'fulltime', 'contract')),
 emp_dept     CHAR(3)       NOT NULL WITH DEFAULT,
 salary       DECIMAL(7,2)  NOT NULL
    CONSTRAINT check_salary
      CHECK (salary < 50000.00),
 commission   DECIMAL(7,2),
 bonus        DECIMAL(7,2)
) IN db.ts;
```

Benefits of Check Constraints

- Enforce business rules directly into the database without requiring additional application logic.
- Non-bypassable
- Improve data integrity
- Promote consistency
- Outperform most other methods of enforcing data integrity

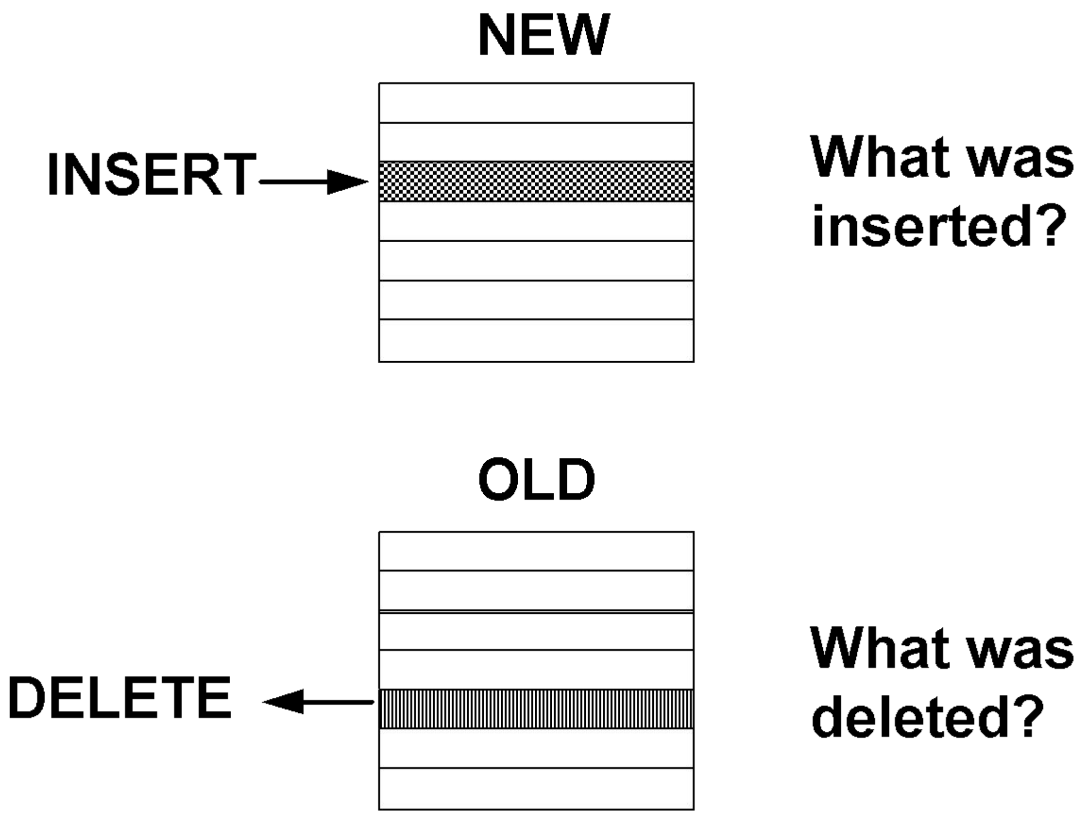
Triggers

- *Triggers* are event-driven specialized procedures that are attached to database tables.
- Execution is automatic and implicit
  - You can not explicitly code a “run trigger” statement
- Trigger is executed, or “fired,” based upon a pre-defined “firing” activity:
  - Database Modification
    - INSERT
    - UPDATE
    - DELETE
- Triggers are quite useful to support:
  - Business rules
  - Redundant data
  - Derived data
  - Data validation

Creating Triggers

- To CREATE a TRIGGER you need:
  - Trigger Name
  - Triggering Table
  - Activation
    - BEFORE / AFTER
  - Triggering Event
    - INSERT, UPDATE, DELETE
  - Granularity
    - FOR EACH ROW / FOR EACH STATEMENT
  - Transition Variables or Table
    - REFERENCING: OLD / NEW
  - Triggered Action - what does the trigger do...

Transition Variables and Tables



Triggered Action

- Each trigger has a triggered action, consisting of two parts:
  - Trigger Condition: When the condition is true, the trigger body is executed.
    - If no trigger condition is coded, the trigger body executes every time the trigger is activated.
  - Trigger Body: The trigger body is the SQL code to be executed when the trigger condition is true.
    - Begins with BEGIN ATOMIC; ends with END.

Trigger Example

```
CREATE TRIGGER NEWHIRE1
 AFTER INSERT
 ON EMPLOYEE
 FOR EACH ROW
 BEGIN ATOMIC
  UPDATE COMPANY_STATS
    SET NBEMP = NBEMP + 1;
 END
```

NOTE
This trigger updates derived data. A counter column is incremented as new employees are inserted.

Another Trigger Example

```
CREATE TRIGGER NEWHIRE2
  AFTER INSERT
  ON EMPLOYEE
  REFERENCING NEW AS N
  FOR EACH ROW
  BEGIN ATOMIC
      UPDATE DEPT_STATS
        SET NBEMP = NBEMP + 1
        WHERE DEPT_ID = N.DEPT_ID;
    END
```

NOTE
A table can have multiple triggers set up for it. This one, like the last one, is on the EMPLOYEE table

NOTE
New data, after the INSERT occurs, can be referenced using the REFERENCING clause.

When Does a Trigger Fire

- Triggers can be coded to fire at two different times:
  - BEFORE the firing activity occurs
    - A “before” trigger executes before the firing activity occurs
  - AFTER the firing activity occurs
    - An “after” trigger executes after the firing activity occurs.

Nested Triggers

- Triggers can contain INSERT, UPDATE, and DELETE statements.
  - Therefore, a data modification fires a trigger that can cause another data modification that fires yet another trigger.
- When a trigger contains INSERT, UPDATE, and/or DELETE logic, the trigger is said to be a *nested trigger*.
  - Most DBMSs, however, place a limit on the number of nested triggers that can be executed within a single firing event.

Trigger Granularity

- A trigger can have statement-level or row-level granularity.
  - A *statement-level trigger* is executed once upon firing, regardless of the actual number of rows inserted, deleted, or updated.
  - A *row-level trigger,* once fired, is executed once for each and every row that is inserted, deleted, or updated.

INSTEAD OF Triggers

- INSTEAD OF triggers can be created to make non-updateable views updateable.
  - Example: a view based on a join of two tables is non-updateable
    - INSTEAD OF triggers can be coded to specify how data is to be inserted, updated, and deleted from such a view
    - After the INSTEAD OF triggers are coded, modifying the view causes the triggers to fire and actually perform the modifications.

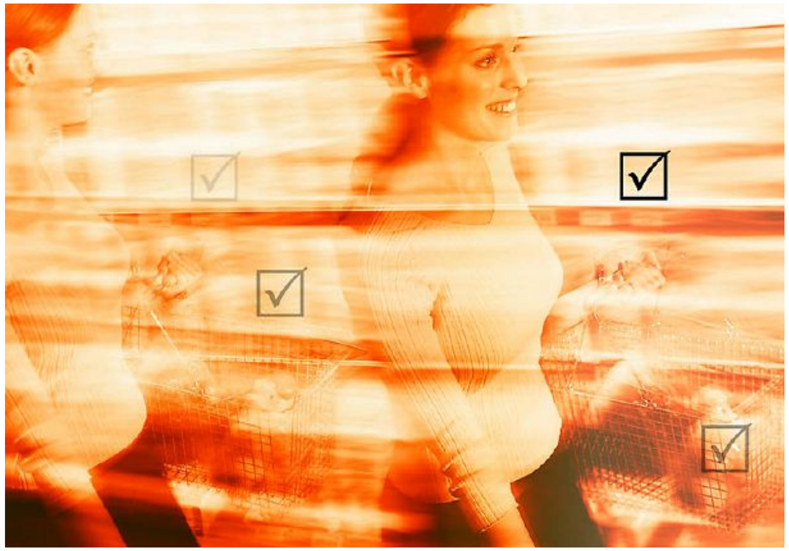
Referential Integrity

- Referential Integrity (RI) is a method for ensuring the “correctness” of data.
- The identification of the primary and foreign keys that constitute a relationship between tables is a component of defining referential integrity.
- RI also requires the definition of rules that dictate how modification of data stored in columns involved in the relationship can be accomplished.

http://craigsmullins.com/adi-db2-1.htm

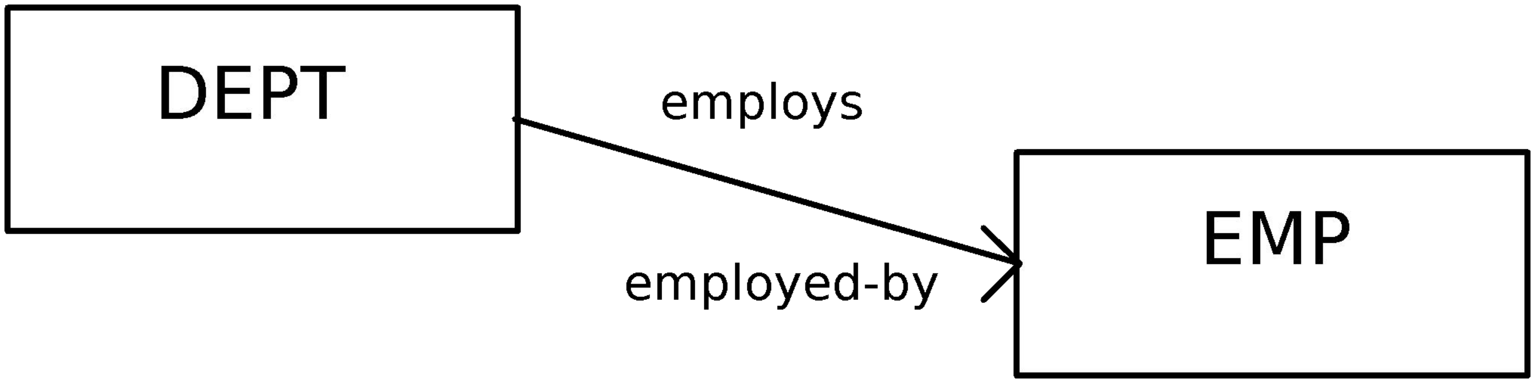
RI in a Nutshell

- RI guarantees that an acceptable value is *always* in the foreign key column.
  - "Acceptable" is defined in terms of an appropriate value as housed in the corresponding primary key, or a null.



RI: Parent / Child

- For any given referential constraint, the *parent table* is the table that contains the primary key, and the *child table* is the table that contains the foreign key.
  - The parent table in the employed-by relationship is the DEPT table.
  - The child table is the EMP table.



RI Rules

Three types of rules can be attached to each referential constraint:

- INSERT rule
- UPDATE rule
- DELETE rule

INSERT Rules

- The *INSERT rule* indicates what will happen if you attempt to insert a value into a foreign key column without a corresponding primary key value in the parent table. There are two aspects to the RI INSERT rule:

1. It is *never* permissible to insert a row into a dependent table with a foreign key value that does not correspond to a primary key value. This is known as the restrict-INSERT rule.
2. Whether actual values *must* be specified instead of nulls.

UPDATE Rule

- The *UPDATE rule* controls updates such that a foreign key value cannot be updated to a value that does not correspond to a primary key value in the parent table.
- There are, however, two ways to view the update rule:
  - Foreign key perspective.
  - Primary key perspective.

UPDATE Rule: FK Perspective

- Once you have assigned a foreign key to a row, either at insertion or afterward, you must decide whether that value can be changed.
- This is determined by looking at the business definition of the relationship and the tables it connects.
- If you permit a foreign key value to be updated, the new value must either be equal to a primary key value currently in the parent table or be null.

UPDATE Rule: PK Perpsective

If a primary key value is updated, three options exist for handling foreign key values:

- *Restricted UPDATE*. The modification of the primary key column(s) is not allowed if foreign key values exist.
- *Neutralizing UPDATE*. All foreign key values equal to the primary key value(s) being modified are set to null. Of course, neutralizing UPDATE requires that nulls be permitted on the foreign key column(s).
- *Cascading UPDATE*. All foreign key columns with a value equal to the primary key value(s) being modified are modified as well.

DELETE Rule

Similar to the primary key perspective of the update rule, three options exist when deleting a row from a parent table:

- *Restricted DELETE*. The deletion of the primary key row is not allowed if a foreign key value exists.
- *Neutralizing DELETE*. All foreign key values equal to the primary key value of the row being deleted are set to null.
- *Cascading DELETE*. All foreign key rows with a value equal to the primary key of the row about to be deleted are deleted as well.

Pendant DELETE

- *Pendant DELETE* is a special type of referential integrity that deals with the treatment of parent table rows when no foreign keys from the child table refer back to the primary key.
- Pendant DELETE RI specifies that the parent table row be deleted after the last foreign key row that refers to it is deleted.
- Pendant DELETE processing cannot be implemented using declarative RI. However, triggers can be used to code the program logic to check for this condition and execute the deletion of the primary key row.

RI Rules Summary

DELETE RESTRICT	If any rows exist in the dependent table, the primary key row in the parent table cannot be deleted.
DELETE CASCADE	If any rows exist in the dependent table, when the parent PK row is deleted, all dependent rows are also deleted.
DELETE NEUTRALIZE	If any rows exist in the dependent table, when the parent PK row is deleted, the foreign key column(s) for all dependent rows are set to NULL as well.
UPDATE RESTRICT	If any rows exist in the dependent table, the primary key column(s) in the parent table cannot be updated.
UPDATE CASCADE	If any rows exist in the dependent table, the primary key column(s) in the parent table are updated, and all foreign key values in the dependent rows are updated to the same value.
UPDATE NEUTRALIZE	If any rows exist in the dependent table, the primary key row in the parent table is deleted, and all foreign key values in the dependent rows are updated to NULL as well.
INSERT RESTRICT	A foreign key value cannot be inserted into the dependent table unless a parent PK value already exists
FOREIGN KEY UPDATE RESTRICTION	A foreign key cannot be updated to a value that does not already exist as a primary key value in the parent table.
PENDANT DELETE	When the last foreign key value in the dependent table is deleted, the parent primary key row is also deleted.

Referential Integrity Guidance

- User-Managed vs. System-Managed RI
  - Use declarative RI instead of program RI
    - performance and ease of use
    - ensure integrity for planned and ad hoc database modification
- Do not use RI for lookup tables
  - consider CHECK constraints vs. lookup tables
- Use triggers only when declarative RI is not workable
  - Triggers are less efficient (usually) than RI
    - but usually better than enforcing in application programs
- Specify indexes on foreign keys

Summary

- No DBMS can ensure the integrity of its data 100% reliably all of the time.
- The DBA must be involved in monitoring and providing for data integrity.
- Both structural and semantic.

Questions