Database Systems -

Introduction to Databases and Data Warehouses

CHAPTER 6 - Database Implementation and Use (Part II Version 2)

IMPLEMENTING USER-DEFINED CONSTRAINTS

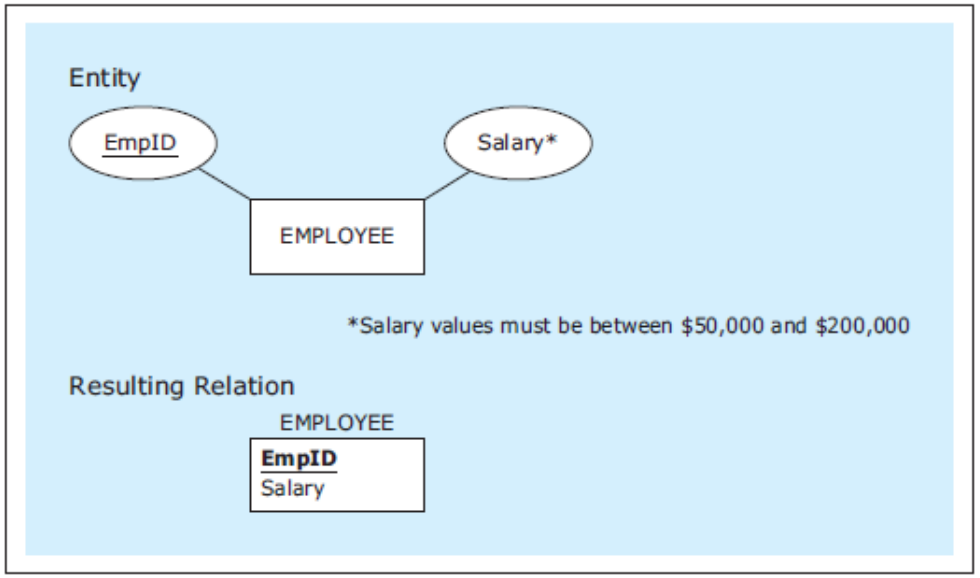
- **Implementing user-defined constraints**
  - Methods for implementing user-defined constraints include:
    - CHECK clause
    - Assertions and triggers
    - Coding in specialized database programming languages that combine SQL with additional non-SQL statements for processing data from databases (such as PL/SQL)
    - Embedding SQL with code written in regular programming languages (such as C++ or Java)

IMPLEMENTING USER-DEFINED CONSTRAINTS

- CHECK
  - Used to specify a constraint on a particular column of a relation

CHECK – Example 1

A relation with a user-defined constraint



SQL code

```
CREATE TABLE employee
(empid   CHAR(4),
salary  NUMBER(6) CHECK (salary >= 50000 AND salary <= 200000),
PRIMARY KEY (empid));

INSERT INTO employee VALUES ('1234', 75000);
INSERT INTO employee VALUES ('2345', 50000);
INSERT INTO employee VALUES ('3456', 55000);
INSERT INTO employee VALUES ('1324', 70000);
INSERT INTO employee VALUES ('9876', 270000);
INSERT INTO employee VALUES ('1010', 30000);
```

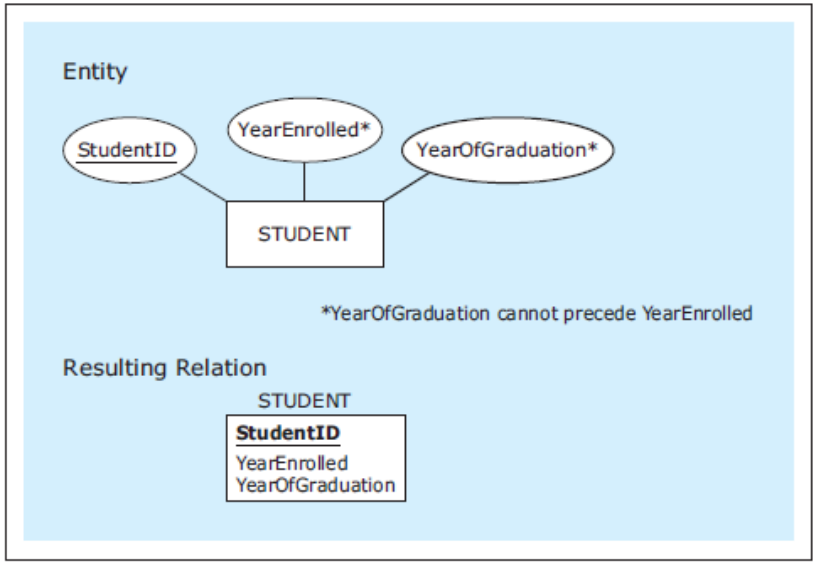
Four inserts accepted, two inserts rejected

EMPLOYEE

EmpID	Salary
1234	75000
2345	50000
3456	55000
1324	70000

CHECK – Example 2

A relation with a user-defined constraint



SQL code

```
CREATE TABLE student
(studentid        CHAR(4),
yearenrolled      INT,
yearofgraduation  INT,
PRIMARY KEY (studentid),
CHECK (yearenrolled <= yearofgraduation));

INSERT INTO student VALUES ('1111', 2012, 2016);
INSERT INTO student VALUES ('2222', 2013, 2017);
INSERT INTO student VALUES ('3333', 2013, 2017);
INSERT INTO student VALUES ('4444', 2013, 2012);
```

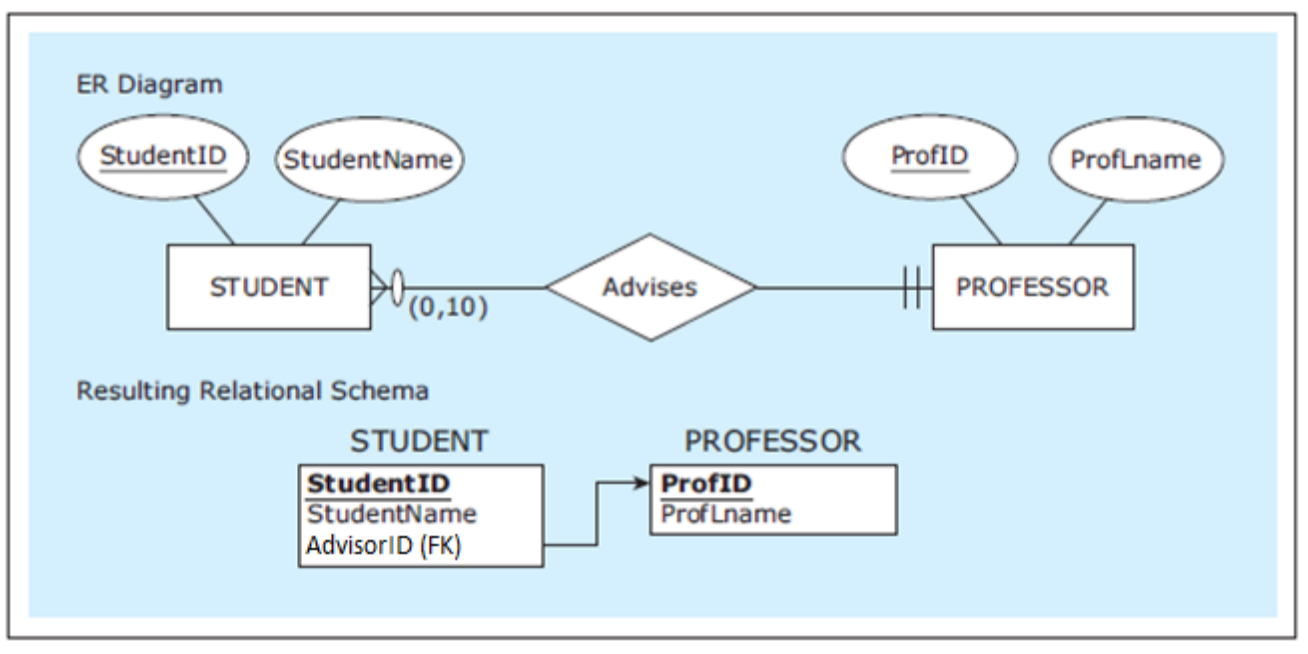
Three inserts accepted, one insert rejected

STUDENT

StudentId	YearEnrolled	YearOfGraduation
1111	2013	2016
2222	2013	2016
3333	2013	2017

ASSERTIONS

- **Assertion:** A mechanism for specifying user-defined constraints
  - Example:




```
CREATE ASSERTION profsadvisingupto10students
    CHECK (
    (SELECT MAX(totaladvised)
     FROM ( SELECT count(*) AS totaladvised
            FROM student
            GROUP BY advisorid)) < 11);
```

TRIGGERS

- Trigger
  - A rule (written using SQL) that is **activated by a deletion** of a record, insertion of a record, or modification (update) of a record in a relation
  - Even though **CREATE ASSERTION is part of the SQL standard**, most RDBMS packages do not implement assertions using CREATE ASSERTION.
  - Most RDBMS packages are capable of implementing the functionality of the assertion through different, more complex types of mechanisms, such as triggers.

Trigger – Example 1

```
CREATE TRIGGER studentinserttrigger
            BEFORE INSERT ON student FOR EACH ROW
      BEGIN
            DECLARE totaladvised INT DEFAULT 0;
            SELECT COUNT(*) INTO totaladvised
            FROM student
            WHERE advisorid = NEW.advisorid;
            IF (totaladvised > 10) THEN
                  SET NEW.advisorid = NULL;
            END IF;
END;
```

Trigger – Example 2

```
CREATE TRIGGER studentupdatetrigger
            BEFORE UPDATE ON student FOR EACH ROW
      BEGIN
            DECLARE totaladvised INT DEFAULT 0;
            SELECT COUNT(*) INTO totaladvised
            FROM student
            WHERE advisorid = NEW.advisorid;
            IF (totaladvised > 10) THEN
                  SET NEW.advisorid = NULL;
            END IF;
END;
```

Trigger – Example (Continued)

```
INSERT INTO student VALUES ('1111', 'Mark', 'P11');
```

EXECUTED AS:

```
INSERT INTO student VALUES ('1111', 'Mark', 'P11');
```

OR when-10 advisee limit is reached:

```
INSERT INTO student VALUES ('1111', 'Mark', null);
```

IMPLEMENTING USER-DEFINED CONSTRAINTS

- **Implementing user-defined constraints**
  - Often implement the logic of user-defined constraints in the front-end database application, but not in the database
  - Important to fully implement user-defined constraints for the proper use of the database
  - From the business use perspective, the enforcement of user-defined constraints is more important than how they are enforced.
    - The choice of implementing user-defined constraints is often based on technical considerations.
      - E.g. whichever runs faster

INDEXING

- INDEX
  - Used for increasing the speed of data search and data retrieval on relations with a large number of records
  - Available in most relational DBMS software tools

Conceptual simplified illustration of the principles on which an index is based

Example relation

CUSTOMER

CustID	CustName	Zip
1000	Zach	60111
1001	Ana	60333
1002	Matt	60222
1003	Lara	60555
1004	Pam	60444
1005	Sally	60555
1006	Bob	60333
1007	Adam	60555
1008	Steve	60222
1009	Pam	60333
1010	Ema	60111
1011	Peter	60666
1012	Fiona	60444

- Data are sorted by CustID.
- CustName: unsorted.

Conceptual simplified illustration of the principles on which an index is based

Linear search – example

Query: Print the record of customer Steve.

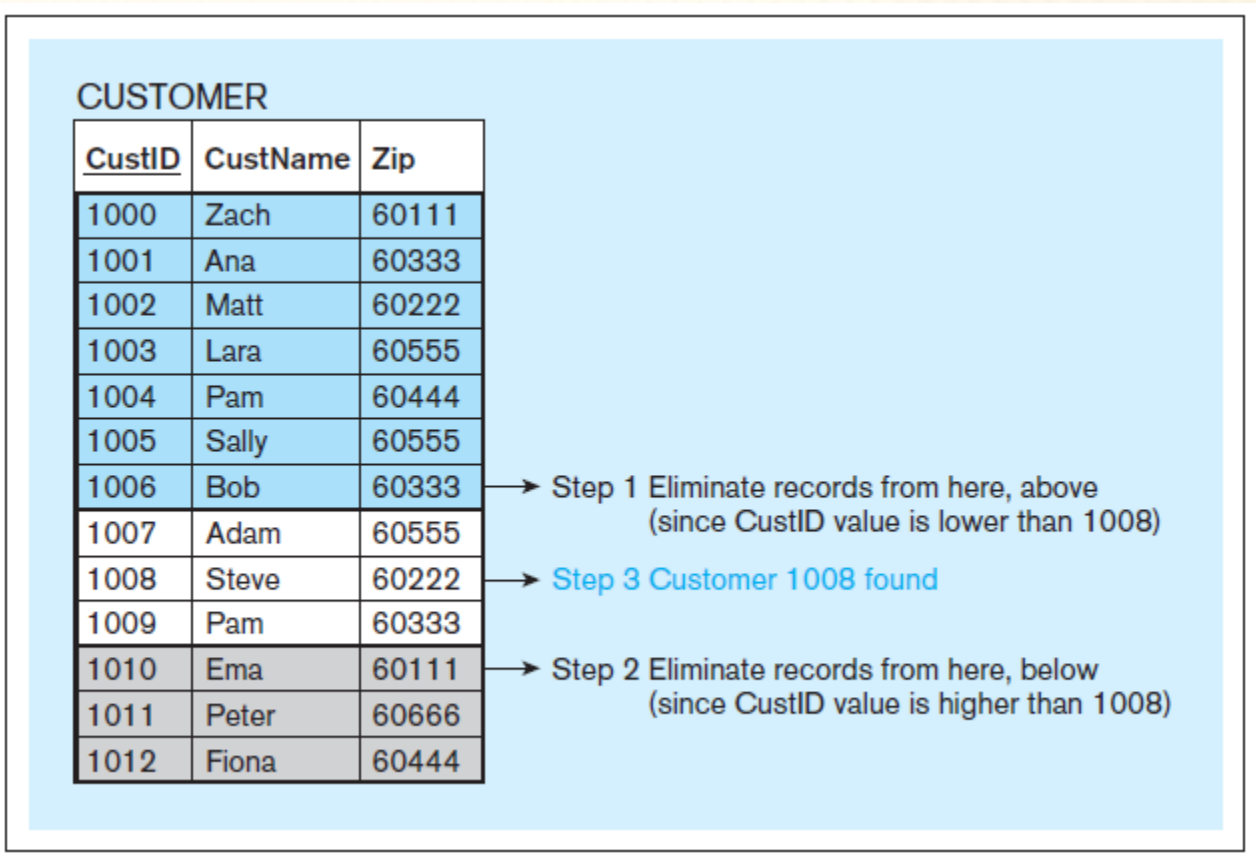
- CustName: unsorted => have to do linear search

CUSTOMER

CustID	CustName	Zip	
1000	Zach	60111	→ Step 1 (not Steve)
1001	Ana	60333	→ Step 2 (not Steve)
1002	Matt	60222	→ Step 3 (not Steve)
1003	Lara	60555	→ Step 4 (not Steve)
1004	Pam	60444	→ Step 5 (not Steve)
1005	Sally	60555	→ Step 6 (not Steve)
1006	Bob	60333	→ Step 7 (not Steve)
1007	Adam	60555	→ Step 8 (not Steve)
1008	Steve	60222	→ Step 9 Customer Steve found
1009	Pam	60333	
1010	Ema	60111	
1011	Peter	60666	
1012	Fiona	60444	

Query: Print the record of customer with ID: 1008.

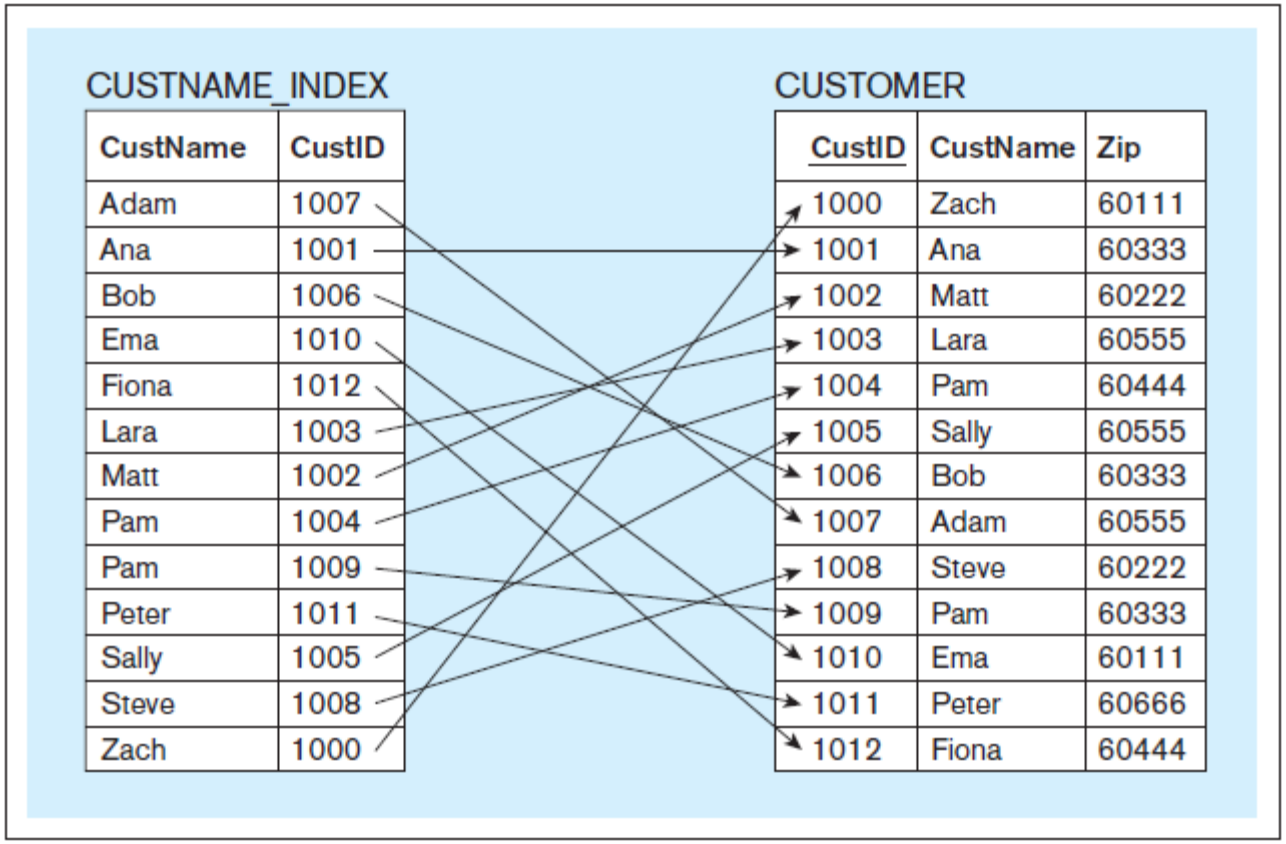
- CustID: sorted => can use binary search (faster)
  - Figure uses: mid = ceil((low+high)/2)



Conceptual simplified illustration of the principles on which an index is based

Index - example

- Create an index table CUSTNAME_INDEX for column CustName
  - Each index entry: (custname value, pointer to related record)
  - Like book index



CUSTNAME_INDEX

CustName	CustID
Adam	1007
Ana	1001
Bob	1006
Ema	1010
Fiona	1012
Lara	1003
Matt	1002
Pam	1004
Pam	1009
Peter	1011
Sally	1005
Steve	1008
Zach	1000

CUSTOMER

CustID	CustName	Zip
1000	Zach	60111
1001	Ana	60333
1002	Matt	60222
1003	Lara	60555
1004	Pam	60444
1005	Sally	60555
1006	Bob	60333
1007	Adam	60555
1008	Steve	60222
1009	Pam	60333
1010	Ema	60111
1011	Peter	60666
1012	Fiona	60444

INDEXING

- Preceding examples
  - Simplified conceptual illustration of the indexing principles
- Different logical and technical approaches used in contemporary RDBMS tools:
  - Clustering indexes
  - Hash indexes
  - B+ trees
  - etc.
- Same goal for all indexing approaches
  - Speed up search and retrieval on the indexed columns
- Cost of using index
  - Slow down data updates involving indexed columns

INDEXING

- CREATE INDEX
  - Example:
    ```
    CREATE INDEX custname_index ON customer(custname);
    ```
  - Once this statement is executed, the effect is that the searches and retrievals involving the CustName column in the relation CUSTOMER are faster
- DROP INDEX
  - Example:
    ```
    DROP INDEX custname_index ON customer(custname);
    ```
  - This statement drops the index, and the index is no longer used