

# 3NF, BCNF and Transaction

Université Grenoble Alpes

18/04/2023

26.04.2023

Bahareh Afshinpour

Main reference: [bahareh.afshinpour@univ-grenoble-alpes.fr](mailto:bahareh.afshinpour@univ-grenoble-alpes.fr)  
*A First Course in Database Systems* (and associated material) by  
J. Ullman and J. Widom, Prentice-Hall

- Still we have some redundancy in our database if our tables are in the 2NF. Because of that redundancy, some anomalies may exist right now.
- The third normal form is an adequate normal form for your database.
- It means if your database is in the 3NF then you can say that it is a good database design.

# What kind of anomaly we have in the 2NF

- The only candidate key is the PassportID (primary-key)
- Since here the candidate key has only one attribute (simple candidate key), then this relation is in the 2NF.

**In the 2NF there should be no partial dependency.**

**What is partial functional dependency (PD) :**

**- A proper subset of candidate key → non-prim attributes(NPA)**

**If only one attribute we have for the candidate key (PassportID) obviously no proper subset is possible. So we will never have a partial dependency. ( This relation is in 2NF)**

PasspostID	Name	Family	country	City	Postal-code	Degree
58459435	David	Mixam	USA	x	9011	1
21476984	Mark	Smith	USA	y	9011	2
74632543	Franck	Thury	USA	z	9011	1
93268547	Sara	Victory	canada	v	91761	3

Redundancy

Update anomaly

-But This relation still has a redundancy

By using postal code we can determine country and city. (postal-code → country,city)



**Non prime attribute → Non prime attribute**

Transitive dependency

NPA  
Is not candidate key or  
partial candidate key

NPA

- The relation is in the 3NF if and only if :

- 1- It is 2NF

- 2- It does not contain any transitive dependency (for non-prime attributes)

Non prime attribute  $\rightarrow$  Non prime attribute

$R(A,B,C,D)$   $FD=\{A \rightarrow B, B \rightarrow C, C \rightarrow D\}$

**Superkey**  
Set of attributes whose closure contains all attributes of a given relation

- Find out candidate key  $ABCD^+ = \{A,B,C,D\}$  We have all the attributes and it is a super key.

- We start to discard it since the candidate key is a minimal super key.  ~~$ABCD^+$~~   $A^+ = \{ABCD\}$  **SK=YES**

CK= Is a superkey whose proper subset is **not** a superkey.

A has no PROPER SUBSET, so **CK=YES**

**Prime attributes (PA)** are those which are part of candidate keys.

**PA=A**

$R(A,B,C,D)$   $FD=\{A \rightarrow B, B \rightarrow C, C \rightarrow D\}$        $PA=A$

**Short trick:** If prime attributes are present on the **right-hand** side of any functional dependency, then there would be more candidate keys. If not, there would be no more candidate keys.

In this example, we have no more prime attributes. The only candidate key is A

**Non prime attribute  $\rightarrow$  Non prime attribute**

If this kind of dependency is present in the relation, this relation is not in the 3NF.

$A \rightarrow B$  , A is the prime attributes - no problem

$B \rightarrow C$  , B and C are the non- prime attributes - transitive dependency 

$C \rightarrow D$  , C and D are the non- prime attributes - transitive dependency 

So this relation is not in the 3NF.

### 3.2.9 Exercises for Section 3.2

**Exercise 3.2.1:** Consider a relation with schema  $R(A, B, C, D)$  and FD's  $AB \rightarrow C$ ,  $C \rightarrow D$ , and  $D \rightarrow A$ .

- a) What are all the nontrivial FD's that follow from the given FD's? You should restrict yourself to FD's with single attributes on the right side.
- b) What are all the keys of  $R$ ?
- c) What are all the superkeys for  $R$  that are not keys?

**Example 3.8:** Let us consider a relation with attributes  $A, B, C, D, E$ , and  $F$ . Suppose that this relation has the FD's  $AB \rightarrow C$ ,  $BC \rightarrow AD$ ,  $D \rightarrow E$ , and  $CF \rightarrow B$ . What is the closure of  $\{A, B\}$ , that is,  $\{A, B\}^+$ ?

# Normalization Example

- Is R in the 3NF?

$R(A,B,C,D,E,F)$      $FD=\{AB \rightarrow CDEF, BD \rightarrow F\}$

$ABCDEF^+ = \{A,B,C,D,E,F\}$

$AB^+ = \{A,B,C,D,E,F\}$      $AB$  is a super key. Check out candidate key :  $A^+ = \{A\}$      $B^+ = \{B\}$

$AB$  is a candidate key so prime= $\{A,B\}$

To check out if more candidate key exist : see right side and find prime attribute

No more candidate key : so candidate key= $\{AB\}$  , Prime= $\{A,B\}$  Non-Prime= $\{C,D,E,F\}$



**Non prime attribute  $\rightarrow$  Non prime attribute**

$FD=\{AB \rightarrow CDEF, BD \rightarrow F\}$      $AB \rightarrow CDEF$  prime  $\rightarrow$  non-prime

$BD \rightarrow F$  B is prime, D is non-prime. BD is Non-prime. Non-prime  $\rightarrow$  Non-prime

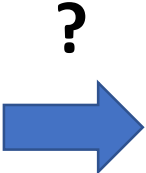
**R is not in 3NF**

# Decomposition

- Suppose a given relation is in the 2NF, we have to divide(decompose) the relation to have a higher normal form.
- For decomposition must follow some properties.
  - Dependency preserving decomposition
  - Lossless join decomposition

- $R(A,B,C)$
- $FD=G$

A	B	C
1	5	7
2	5	2
3	9	7
4	9	2



A	B	B	C

R1 R2 R3 R4 ..... Rn  
 F1 U F2 U F3 U F4 U ..... F11 → F=G



# Example of Dependency preserving

- Suppose we divide R into two sub relations R1 and R2. Is this dependency preserving decompositions?

$R(A,B,C,D,E)$  FD=  $\{A \rightarrow B, B \rightarrow C, C \rightarrow D, D \rightarrow A\}$

$R1(A,B,C)$   $R2(C,D,E)$

First, we need to find the FD for R1 and R2.

- Find out the closure of all the attributes in the relation.

For example for  $R1(A,B,C)$ , one member, two member, ... :

We do not have D in R1

$A^+ = \{A, B, C, D\}$	$\rightarrow$	$A \rightarrow BC$
$B^+ = \{B, C, D, A\}$	$\rightarrow$	$B \rightarrow CA$
$C^+ = \{C, D, A, B\}$	$\rightarrow$	$C \rightarrow AB$
$AB^+ = \{A, B, C, D\}$	$\rightarrow$	<del><math>AB \rightarrow C</math></del> But it is a duplicate FD Since only using $A \rightarrow BC$ means $A \rightarrow C$ so $AB \rightarrow C$ . So we are not getting sth new

Second,  $R2(C,D,E)$ , we find closure of each single attributes

$C^+ = \{C, D, A, B\}$	$C \rightarrow D$	
$D^+ = \{D, A, B, C\}$	$D \rightarrow C$	
$E^+ = \{E\}$		
$CD^+ = \{CDAB\}$		
$DE^+ = \{DE, AB, C\}$	<del><math>DE \rightarrow C</math></del>	It would be duplicate because we have $D \rightarrow C$
$CE^+ = \{CEDAB\}$	<del><math>CE \rightarrow D</math></del>	It would be duplicate because we have $C \rightarrow D$

$F1 = \{A \rightarrow BC, B \rightarrow CA, C \rightarrow AB\}$

$F2 = \{D \rightarrow C, C \rightarrow D\}$

$F1 \cup F2 = \{A \rightarrow BC, B \rightarrow CA, C \rightarrow AB, D \rightarrow C, C \rightarrow D\}$

- If every FD of F is a member of G and every FD of G is a member of F then we can say they are equivalent.

**F1 U F2 == G ????**

G = {A → B, B → C, C → D, D → A}

F1 U F2 = F = {A → BC, B → CA, C → AB, D → C, C → D}

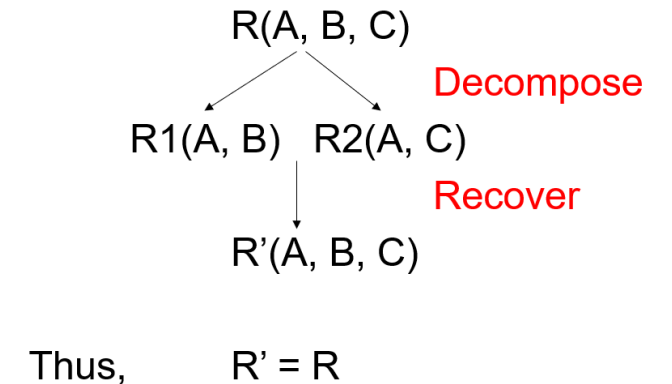
D<sup>+</sup> = {DCAB}

It means D → A is in the F

Yes. This example is a dependency-preserving decomposition

# Lossless join decomposition

- Decomposition of a relation  $R$  is lossless if it is feasible to reconstruct the relation from decomposed sub relations by using joints.



A decomposition  $\{R1, R2, \dots, Rn\}$  of a relation  $R$  is called a **lossless decomposition** for  $R$  if the natural join of  $R1, R2, \dots, Rn$  produces exactly the relation  $R$ .

A	B	C
---	---	---

$\alpha$	1	A
$\beta$	2	B

$r$

A	B
---	---

$\alpha$	1
$\beta$	2

$\Pi_{A,B}(r)$

B	C
---	---

1	A
2	B

$\Pi_{B,C}(r)$

$\Pi_A(r) \bowtie \Pi_B(r)$

A	B	C
---	---	---

$\alpha$	1	A
$\beta$	2	B

# Normalization Example

**Example 3.8:** Let us consider a relation with attributes  $A, B, C, D, E,$  and  $F$ . Suppose that this relation has the FD's  $AB \rightarrow C, BC \rightarrow AD, D \rightarrow E,$  and  $CF \rightarrow B$ . What is the closure of  $\{A, B\}$ , that is,  $\{A, B\}^+$ ?

First, split  $BC \rightarrow AD$  into  $BC \rightarrow A$  and  $BC \rightarrow D$ . Then, start with  $X = \{A, B\}$ . First, notice that both attributes on the left side of FD  $AB \rightarrow C$  are in  $X$ , so we may add the attribute  $C$ , which is on the right side of that FD. Thus, after one iteration of Step 3,  $X$  becomes  $\{A, B, C\}$ .

Next, we see that the left sides of  $BC \rightarrow A$  and  $BC \rightarrow D$  are now contained in  $X$ , so we may add to  $X$  the attributes  $A$  and  $D$ .  $A$  is already there, but  $D$  is not, so  $X$  next becomes  $\{A, B, C, D\}$ . At this point, we may use the FD  $D \rightarrow E$  to add  $E$  to  $X$ , which is now  $\{A, B, C, D, E\}$ . No more changes to  $X$  are possible. In particular, the FD  $CF \rightarrow B$  can not be used, because its left side never becomes contained in  $X$ . Thus,  $\{A, B\}^+ = \{A, B, C, D, E\}$ .  $\square$

# BCNF(Boyce-Codd Normal Form)

- It is strong version of 3NF.

The relation is in BCNF:

- It is in 3NF
- For each non-trivial functional dependency  
The **left** hand side of dependency must be a Supekey ( **X**->Y )

# Example

- $R(A,B,C)$  FD={ $A \rightarrow B$ ,  $B \rightarrow C$ ,  $C \rightarrow A$ }

First find out the candidate key:

$ABC^+ = \{A,B,C\}$  we have all the relation so it is a super key

We start to discard as many attribute that we can since a candidate key is a minimal super key.

~~$ABC$~~  $^+ = \{A,B,C\}$      $A^+ = \{A,B,C\}$     **A has no proper subset. If no proper subset is possible, then there is no chance to have superkey**    CK= yes, Prime attribute={A}

More candidate key ??? Check the right side of the FD.

If you can find the prime attribute we have more candidate key

**$C \rightarrow A$  so here we have more candidate key**

# Example

$R(A,B,C)$   $FD=\{A \rightarrow B, B \rightarrow C, C \rightarrow A\}$

**$C \rightarrow A$  so here we have more candidate key**

$CK=A$  , replace A with C. Is C candidate key?? We have to check.

$C^+ = \{C,A,B\}$  and C has no proper subset.  $CK=yes$

Prime attributes= $\{A,C\}$

see right side. We have  $B \rightarrow C$  . So we replace C by B. ....

Prime attributes= $\{A,C,B\}$

See the left side:

$A \rightarrow B$

$B \rightarrow C$

$C \rightarrow A$

All the left side are candidate key. Definitely they are super key. So

**This relation is in the BCNF**



# Find the highest normal form in R

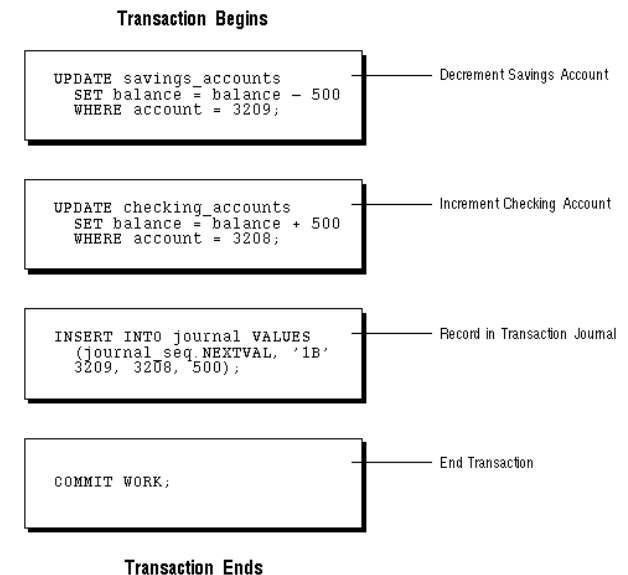
- $R(A,B,C,D,E)$   $FD=\{A \rightarrow BCDE, BC \rightarrow ACE, D \rightarrow E\}$

Transaction


# Transaction

- Transaction is a group or set of tasks into a single execution unit.
- Each transaction begins with a specific task and ends when all the tasks in the group successfully complete.
- If any of tasks fails the transactions fails.
- The effects of all the SQL statements in a transaction can be either all *committed* (applied to the database) or all *rolled back* (undone from the database).

- Decrement the savings account
- Increment the checking account
- Record the transaction in the transaction journal



# Transaction

Unit of works  Tx

In this unit of work, there could be multiple changes on **multiple different rows** in **different tables** that occur all that once.

For example 

Insert Into Table1  
Update Table 2

A transaction is a logical, atomic unit of work that contains one or more SQL statements.

An important trait (property) of a transaction is the fact that these two things must either succeed or fail together as a unit  
**Therefore, a transactions has only two results : - success -Failure**

**Example 6.41:** Let us picture another common sort of database: a bank's account records. We can represent the situation by a relation

`Accounts(acctNo, balance)`

Consider the operation of transferring \$100 from the account numbered 123 to the account 456. We might first check whether there is at least \$100 in account 123, and if so, we execute the following two steps:

1. Add \$100 to account 456 by the SQL update statement:

```
UPDATE Accounts
SET balance = balance + 100
WHERE acctNo = 456;
```

2. Subtract \$100 from account 123 by the SQL update statement:

```
UPDATE Accounts
SET balance = balance - 100
WHERE acctNo = 123;
```

Now, consider what happens if there is a failure after Step (1) but before Step (2). Perhaps the computer fails, or the network connecting the database to the processor that is actually performing the transfer fails. Then the database is left in a state where money has been transferred into the second account, but the money has not been taken out of the first account. The bank has in effect given away the amount of money that was to be transferred. □

- When using the generic SQL interface, each statement is a transaction by itself.
- Also, SQL allows the programmer to group several statement into a single transaction.
- The SQL command **START TRANSACTION** is used to mark the beginning of the transaction.
- There are two ways to end a transaction:
  - Using COMMIT
  - Using ROLLBACK

# Example

We have two different users.

```
SQL> select * from department;

DEPTID DEPTNAME
-----
10 BCA
20 MCA
30 BBA
40 MSc
50 MTech
70 Btech
80 MTechCS

7 rows selected.
```

```
SQL> select * from system.department;

DEPTID DEPTNAME
-----
10 BCA
20 MCA
30 BBA
40 MSc
50 MTech
70 Btech
80 MTechCS

7 rows selected.
```

User1(system) want to add one new tuple in the department

# Example

```
SQL> insert into department values(90,'MTechIT');
1 row created.
SQL> select * from department;
DEPTID DEPTNAME
-----
10 BCA
20 MCA
30 BBA
40 MSc
50 MTech
70 Btech
80 MTechCS
90 MTechIT
8 rows selected.
```

If we proceed in the sys terminal, and check the department

```
SQL> select * from system.department;
DEPTID DEPTNAME
-----
10 BCA
20 MCA
30 BBA
40 MSc
50 MTech
70 Btech
80 MTechCS
7 rows selected.
```

We can not see any changes in the table

The changes must be saved otherwise it is discarded

When any user is updating any changes, only the user itself can see the changes without commit or rollback. No other user can access or view the updates, as it is not permanently saved by the user who performed them.



# Example

```
SQL> select * from department;
```

DEPTID	DEPTNAME
--------	----------

10	BCA
20	MCA
30	BBA
40	MSc
50	MTech
70	Btech
80	MTechCS
90	MTechIT

8 rows selected.

```
SQL> commit;
```

Commit complete.

```
SQL> select * from system.department;
```

DEPTID	DEPTNAME
--------	----------

10	BCA
20	MCA
30	BBA
40	MSc
50	MTech
70	Btech
80	MTechCS
90	MTechIT

8 rows selected.

# ROLLBACK

```
SQL> delete from department;
```

```
8 rows deleted.
```

```
SQL> select * from department;
```

```
no rows selected
```

```
SQL> rollback;
```

```
Rollback complete.
```

```
SQL> select * from department;
```

```
DEPTID DEPTNAME
```

```
-----  
10 BCA
```

```
20 MCA
```

```
30 BBA
```

```
40 MSc
```

```
50 MTech
```

```
70 Btech
```

```
80 MTechCS
```

```
90 MTechIT
```

```
8 rows selected.
```

- Without Commit or Rollback, the permanent update is not possible.
- But if the user changes or updates sth and disconnects from the database properly, commits occur.

If user disconnects from the database after some changes auto commits occurs.

# Transaction

- **START TRANSACTION** or **BEGIN** start a new transaction.
- **COMMIT** commits the current transaction, making its changes permanent.
- **ROLLBACK** rolls back the current transaction, canceling its changes.
- **SET autocommit** disables or enables the default autocommit mode for the current session.

```
--Begin the transaction  
SET TRANSACTION READ WRITE;
```

```
--Create the new project  
INSERT INTO project  
  SELECT 1007, project_name, project_budget FROM project  
  WHERE project_id = 1002;
```

```
--Point the time log rows in project_hours to the new project number  
UPDATE project_hours  
  SET project_id = 1007  
  WHERE project_id = 1002;
```

```
--Delete the original project record  
DELETE FROM project  
  WHERE project_id=1002;
```

```
COMMIT;
```

**SET TRANSACTION** marks the beginning of a transaction. Any changes you make to your data following the beginning of a transaction are not made permanent until you issue a **COMMIT**.

Tip

Using **SET TRANSACTION** to begin a transaction is **optional**. A new transaction begins implicitly with the first DML statement that you execute after you make a database connection or with the first DML statement that you execute following a **COMMIT** or a **ROLLBACK** (or any DDL statement such as **TRUNCATE**). You need to use **SET TRANSACTION** only when you want transaction attributes such as **READ ONLY** that are not the default.

**read/write transaction** : Such a transaction is the default, and it allows you to issue statements such as UPDATE and DELETE.

You can also create **read-only transactions**:

If we tell the SQL execution system that our current transaction is *read-only*, that is, it will never change the database, then it is quite possible that the SQL system will be able to take advantage of that knowledge. Generally it will be possible for many read-only transactions accessing the same data to run in parallel, while they would not be allowed to run in parallel with a transaction that wrote the same data.

We tell the SQL system that the next transaction is read-only by:

```
SET TRANSACTION READ ONLY;
```

This statement must be executed before the transaction begins.

# ACID Transactions

- A DBMS is expected to support “*ACID transactions*,” processes that are:
  - *Atomic* : All actions of a transaction are atomic and either they are all performed or none of the actions are performed.
  - *Consistent* : Each transaction, when run alone, must preserve the consistency of the database.
  - *Isolated* : Each transaction is isolated (protected) from the effects of other concurrently running transactions.
  - *Durable* : Effects of a process do not get lost if the system crashes. once a transaction commits, the data should persist in the database even if the system crashes before the data is written to non-volatile storage.

# Isolation Levels

- SQL defines four *isolation levels* = choices about what interactions are allowed by transactions that execute at about the same time.
- How a DBMS implements these isolation levels is highly complex, and a typical DBMS provides its own options.

# Choosing the Isolation Level

- Within a transaction, we can say:  
SET TRANSACTION ISOLATION LEVEL X

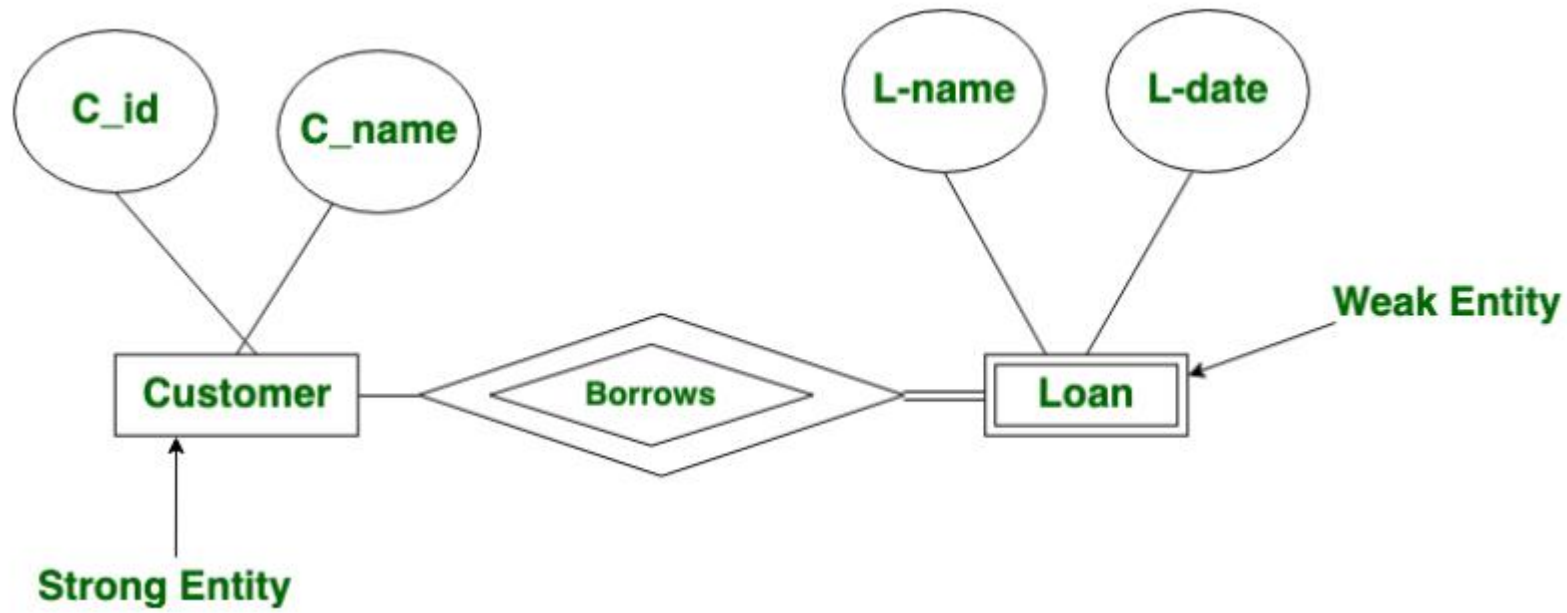
where X =

1. SERIALIZABLE
2. REPEATABLE READ
3. READ COMMITTED
4. READ UNCOMMITTED



```
DECLARE
    <variable declaration>
BEGIN
    <simple sql statement>
    <simple sql statement>
    <simple sql statement>
    SAVEPOINT do_insert;
    <sql insert statement>
EXCEPTION
    WHEN DUP_VAL_ON_INDEX THEN
        ROLLBACK TO do_insert;
        DBMS_OUTPUT.PUT_LINE('Insert has been rolled back');
END;
--and commit outside the transaction
```

SQL



# Updates

- To change certain attributes in certain tuples of a relation:

```
UPDATE <relation>
```

```
SET <list of attribute assignments>
```

```
WHERE <condition on tuples>;
```

Change drinker Fred's phone number to 555-1212:

```
UPDATE Drinkers  
SET phone = '555-1212'  
WHERE name = 'Fred';
```

# Example: Update Several Tuples

- Make \$4 the maximum price for beer:

```
UPDATE Sells
```

```
SET price = 4.00
```

```
WHERE price > 4.00;
```

# Adding Attributes

- We may add a new attribute (“column”) to a relation schema by:

```
ALTER TABLE <name> ADD  
    <attribute declaration>;
```

- Example:

```
ALTER TABLE Bars ADD phone CHAR(16)
```

# Deleting Attributes

- Remove an attribute from a relation schema by:

**ALTER TABLE <name>**

**DROP <attribute>;**

- Example: we don't really need the license attribute for bars:

```
ALTER TABLE Bars DROP license;
```

# Views

- A *view* is a “virtual table” = a relation defined in terms of the contents of other tables and views.
  - $V = \text{viewquery}(R_1, R_2, \dots, R_N)$
- Declare by:  
**CREATE VIEW <name> AS <query>;**



# Example: View Definition

- `CanDrink(drinker, beer)` is a view “containing” the drinker-beer pairs such that the drinker frequents at least one bar that serves the beer:

```
CREATE VIEW CanDrink AS
  SELECT drinker, beer
  FROM Frequents, Sells
  WHERE Frequents.bar = Sells.bar;
```

# Example: Accessing a View

- a limited ability to modify views if it makes sense as a modification of one underlying base table.
- Example query:

```
SELECT beer FROM CanDrink  
WHERE drinker = 'Sally';
```

# Query + Subquery Solution

```
SELECT bar  
FROM Sells  
WHERE beer = 'Miller' AND
```

```
price = (SELECT price  
        FROM Sells  
        WHERE bar = 'Joe''s Bar'  
        AND beer = 'Bud');
```

The price at  
which Joe  
sells Bud



# The IN Operator

- `<tuple> IN <relation>` is true if and only if the tuple is a member of the relation.
  - `<tuple> NOT IN <relation>` means the opposite.
- IN-expressions can appear in WHERE clauses.
- The `<relation>` is often a subquery.

# Example

- From **Beers(name, manf)** and **Likes(drinker, beer)**, find the name and manufacturer of each beer that Fred likes.

```
SELECT *
```

```
FROM Beers
```

```
WHERE name IN (SELECT beer
```

The set of  
beers Fred  
likes



```
FROM Likes
```

```
WHERE drinker = 'Fred');
```

# Constraints

- A *constraint* is a relationship among data elements that the DBMS is required to enforce.
- Several kind of constraints
  - Example: primary key constraints.

# Most used kinds of constraints

- **Keys**
  - Foreign-key, or referential integrity.
- **Value-based** constraints.
  - Constrain values of a particular attribute.

# Actions Taken

- Suppose  $R = \text{Sells}$ ,  $S = \text{Beers}$ .
- An insert or update to Sells that introduces a nonexistent beer must be rejected.
- A deletion or update to Beers that removes a beer value found in some tuples of Sells can be handled in three ways (next slide).



# Actions Taken

1. **Default** : Reject the modification.
2. **Cascade** : Make the same changes in Sells.
  - Deleted beer: delete Sells tuple.
  - Updated beer: change value in Sells.
3. **Set NULL** : Change the beer to NULL.

# Example: Cascade

- Delete the Bud tuple from Beers:
  - Then delete all tuples from Sells that have beer = 'Bud'.
- Update the Bud tuple by changing 'Bud' to 'Budweiser':
  - Then change all Sells tuples with beer = 'Bud' so that beer = 'Budweiser'.

# Example: Set NULL

- Delete the Bud tuple from Beers:
  - Change all tuples of Sells that have beer = 'Bud' to have beer = NULL.
- Update the Bud tuple by changing 'Bud' to 'Budweiser':
  - Same change.