Constraints and SQL

Seminar Session

(Monday, Sept. 14, 2015)

AGENDA

- **Sample Database Practice**
- Constraints
  - Primary key declarations (already covered).
  - Foreign-keys = referential integrity constraints.
  - Attribute- and tuple-based checks = constraints within relations.
  - SQL Assertions = global constraints.
    - Not found in Oracle.
  - Oracle Triggers.
    - A substitute for assertions.

Sample Database: Practice

- Consider the following schema:

  Book (isbn, title, author, subject_id)
  Authors (id, last_name, first_name)
  Publishers (id, name, address)
  Publishes (book_id, publisher_id)
  Copy (ISBN, copy_id, shelf_no, position)
  Readers (id, last_name, first_name)
  Borrow (isbn, copy_id, reader_id, return_date)

- **Query1:** Sort the publishers in descending order according to the number of books that they have published

```
SELECT publishers.name, count(*) AS cnt
FROM publishes, publishers
WHERE publishers.id = publishes.publisher_id
GROUP BY (publishers.name)
ORDER BY cnt DESC;
```

Sample Database: Practice…

- Consider the following schema:

  Book (isbn, title, author, subject_id)
  Authors (id, last_name, first_name)
  Publishers (id, name, address)
  Publishes (book_id, publisher_id)
  Copy (ISBN, copy_id, shelf_no, position)
  Readers (id, last_name, first_name)
  Borrow (isbn, copy_id, reader_id, return_date)

- **Query2:** List all the titles of the books that have been borrowed

```
SELECT DISTINCT title
FROM books, borrow
WHERE books.isbn = borrow.isbn
```

Sample Database: Practice…

- Consider the following schema:

  Book (isbn, title, author, subject_id)
  Authors (id, last_name, first_name)
  Publishers (id, name, address)
  Publishes (book_id, publisher_id)
  Copy (ISBN, copy_id, shelf_no, position)
  Readers (id, last_name, first_name)
  Borrow (isbn, copy_id, reader_id, return_date)

- **Query3:** Find the first name of the readers who borrowed the most number of books

```
WITH aggr AS
        (SELECT reader_id AS rdr, count(*) AS cnt FROM
        borrow GROUP BY borrow.reader_id)
SELECT id, first_name
FROM readers, aggr
WHERE readers.id=aggr.rdr AND
        aggr.cnt IN (SELECT max(cnt) FROM aggr)
```

Sample Database: Practice…

- Consider the following schema:

  Book (isbn, title, author, subject_id)
  Authors (id, last_name, first_name)
  Publishers (id, name, address)
  Publishes (book_id, publisher_id)
  Copy (ISBN, copy_id, shelf_no, position)
  Readers (id, last_name, first_name)
  Borrow (isbn, copy_id, reader_id, return_date)

- **Query4:** Find the most popular author

```
WITH at_id AS
        (SELECT author_id, count(*) AS cnt FROM books, copies, borrow
        WHERE books.isbn = copies.isbn AND copies.isbn = borrow.isbn AND
        copies.copy_id = borrow.copy_id GROUP BY author_id)
SELECT * FROM at_id, authors
WHERE authors.id = at_id.author_id AND
        at_id.cnt IN (SELECT max(cnt) FROM at_id)
```

Sample Database: Practice…

- Consider the following schema:

  Book (isbn, title, author, subject_id)
  Authors (id, last_name, first_name)
  Publishers (id, name, address)
  Publishes (book_id, publisher_id)
  Copy (ISBN, copy_id, shelf_no, position)
  Readers (id, last_name, first_name)
  Borrow (isbn, copy_id, reader_id, return_date)

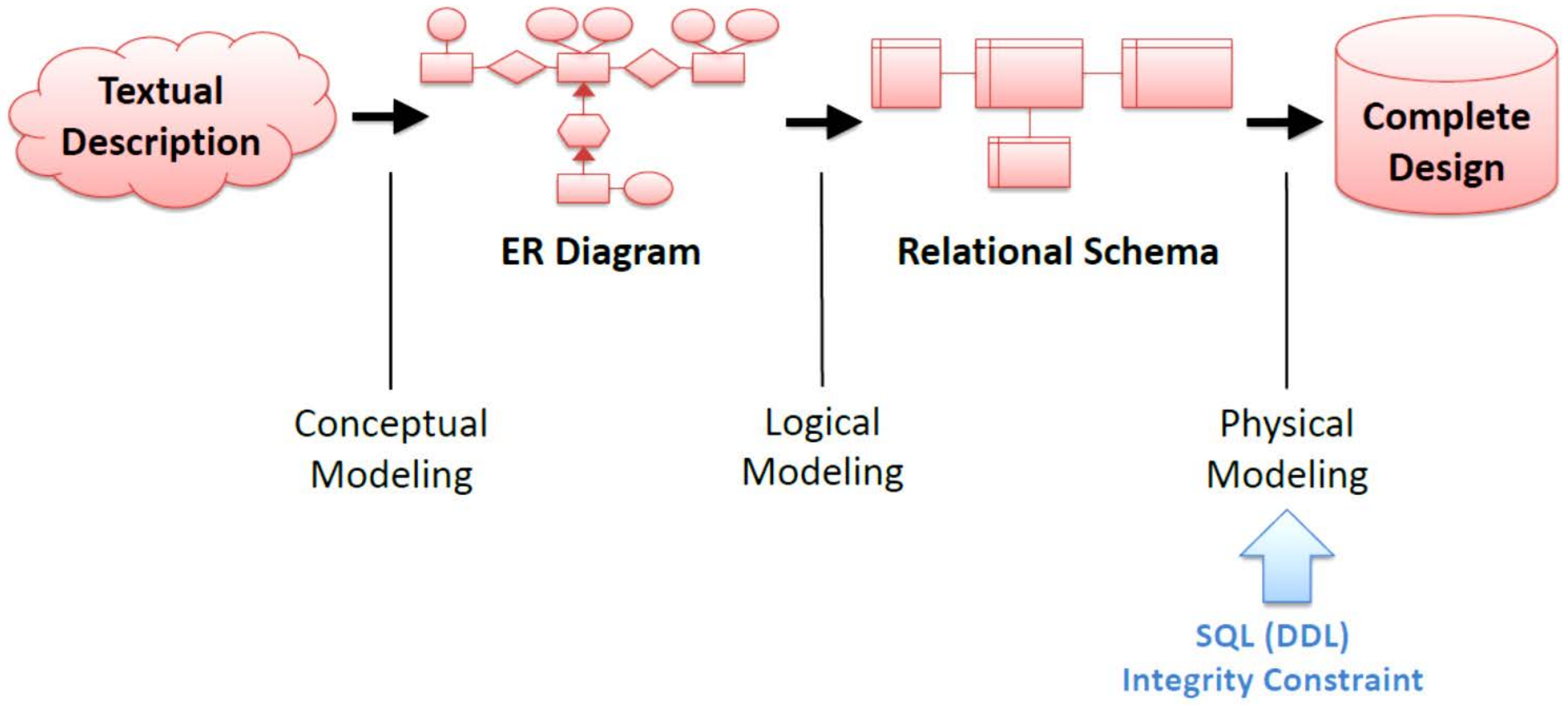
- **Query5:** Find the books that are overdue('2015-09-14') and list the book title, return date, reader first- and lastname

```
SELECT books.title, return_date, readers.first_name, readers.last_name
FROM borrow, copies, books, readers
WHERE books.isbn = copies.isbn AND
          copies.isbn = borrow.isbn AND
          copies.copy_id = borrow.copy_id AND
          readers.id = borrow.reader_id AND
          return_date < '2015-09-14';
```

AGENDA

- Sample Database Practice
- **Constraints**
  - Primary key declarations (already covered).
  - Foreign-keys = referential integrity constraints.
  - Attribute- and tuple-based checks = constraints within relations.
  - SQL Assertions = global constraints.
    - Not found in Oracle.
  - Oracle Triggers.
    - A substitute for assertions.

Database Design



Integrity Constraints

- ACID Principle in DBMS:
  - **Atomicity:** (commit) all or nothing
  - **Consistency:** all tuples in DB (=states) must respect the defined rules
  - **Isolation:** transaction updates do not interfere with each other (not updating the same tuple)
  - **Durability:** data is stored permanently once committed
- This is all about Consistency!
  - Any transaction will bring DB from on valid state to another, i.e. any data written to the database must be valid according to all defined rules.

Integrity Constraints…

- How can we define rules in a DB?
  1. **Attribute Constraints**: intra-table rules
  2. **Referential Constraints**: inter-table rules
  3. **Triggers**: more complex rules
- Constraints are used to limit the type of data that can be inserted into a table
- There exist two ways to specify constraints on a relation:
  1. When a table is created: CREATE TABLE
  2. After the table is created: ALTER TABLE

Integrity Constraints…

Name	Description
NOT NULL	This enforces a field to always contain a value.
UNIQUE	This constraint uniquely identifies each record in a table.
PRIMARY KEY	A primary key uniquely identifies each record in a table (= UNIQUE + NOT NULL), each table can have only one primary key.
FOREIGN KEY	A foreign key is a reference that points from one table to the primary key of another table. It prevents actions that would destroy links between tables. [Referential Integrity Constraints]
CHECK	This constraint is used to limit the value range that can be placed in a column. CHECK can limit the values in certain columns based on values in other columns
DEFAULT	It is used to insert a default value into a column. The default value will be added to all new records, if no other value is specified.

Intra-Table Constraints

- Example

```
CREATE TABLE User (
id int(11) PRIMARY KEY,
group varchar(10) DEFAULT ‘guest',
username varchar(20) UNIQUE,
password varchar(30) NOT NULL,
email varchar(30) CHECK (email LIKE ‘%@%’),
CONSTRAINT chk_group CHECK group IN (‘user’,
‘admin’, ‘guest’)
)

ALTER TABLE User ADD
CONSTRAINT uni_email UNIQUE
(email)
```

Inter-Table Constraint: Foreign Keys

In relation *R* a clause that “attribute *A* references *S*(*B*)” says that whatever values appear in the *A* column of *R* must also appear in the *B* column of relation *S*.

- *B* must be declared the primary key for *S*.

Example

```
CREATE TABLE Beers (
   name CHAR(20) PRIMARY KEY,
   manf CHAR(20)
);

CREATE TABLE Sells (
   bar CHAR(20),
   beer CHAR(20) REFERENCES Beers(name),
   price REAL
);
```

Foreign Keys…

- Alternative: add another element declaring the foreign key, as:

```
CREATE TABLE Sells (
    bar CHAR(20),
    beer CHAR(20),
    price REAL,
    FOREIGN KEY beer REFERENCES
                Beers(name)
);
```

- Extra element essential if the foreign key is more than one attribute.

Inter-Table Constraint: Foreign Keys

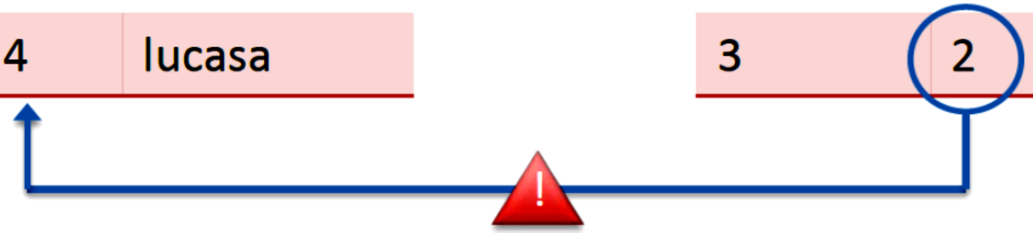
- Is there anything wrong here?

uid	username
1	jribon
3	prtrem
4	lucasa

cid	uid	comment
1	1	Hello World!
2	1	Students are smart.
3	2	I love DB!

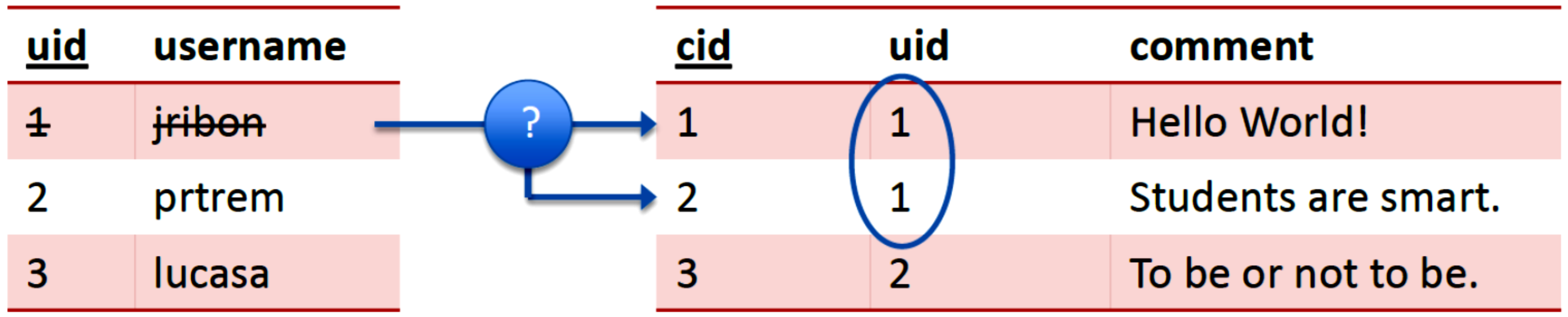
uid	username
1	jribon
3	prtrem
4	lucasa

cid	uid	comment
1	1	Hello World!
2	1	Students are smart.
3	2	I love DB!



Referential Actions

- Referential actions ensure data integrity if a row (in a referenced table) is deleted or updated.
- This defines what should happens if we want to delete (or update) a row that is still referenced in another table.



uid	username
~~1~~	~~jribon~~
2	prtrem
3	lucasa

Referenced Table

cid	uid	comment
1	1	Hello World!
2	1	Students are smart.
3	2	To be or not to be.

Referencing Table

Referential Actions…

- SQL-2003 specifies 5 different referential actions:

Name	Description
CASCADE	The respective rows of the referencing table are deleted or updated together with the referenced row.
RESTRICT	A row cannot be deleted or updated as long as there is a reference to it from a referencing table. Integrity constraint is checked before executing the statement. [Pessimistic: expect violation]
NO ACTION	As RESTRICT, but integrity constraint is checked at the end of the UPDATE or DELETE statement. [Optimistic: assume no violation]
SET NULL	The foreign key value (in the referencing row) is set to NULL when the referenced row is updated or deleted. Note: foreign key must be null-able.
SET DEFAULT	Similar to SET NULL, the foreign key is set to the column's default value when the referenced row is updated or deleted.

Referential Actions: Examples

- Consider the following relational schema:

  Reader (**<u>readerId</u>**, rstName, lastName, address, city, dateOfBirth)
  Book (**<u>isbn</u>**, title, author, numberOfPages, yearOfPublication, publisherName)
  Publisher (**<u>publisherName</u>**, placeOfPublication)
  Categories (**<u>categoryName</u>**, includedIn)
  Copy (**<u>ISBN</u>**, **<u>copyNumber</u>**, shelf, position)
  Loan (**<u>readerId</u>**, **<u>ISBN</u>**, copyNumber, returnDate)
  BookCategory (**<u>ISBN</u>**, **<u>categoryName</u>**)

- Impact of deleting a Publisher?
  - If no book references the publisher → delete
  - If books references the publisher → stop action (or set to NULL)

Referential Actions: Examples

- Consider the following relational schema:

  Reader (<u>**readerId**</u>, rstName, lastName, address, city, dateOfBirth)
  Book (<u>**isbn**</u>, title, author, numberOfPages, yearOfPublication, publisherName)
  Publisher (<u>**publisherName**</u>, placeOfPublication)
  Categories (<u>**categoryName**</u>, includedIn)
  Copy (<u>**ISBN**</u>, <u>**copyNumber**</u>, shelf, position)
  Loan (<u>**readerId**</u>, <u>**ISBN**</u>, copyNumber, returnDate)
  BookCategory (<u>**ISBN**</u>, <u>**categoryName**</u>)

- Impact of deleting a Publisher?
  - If no book references the publisher → delete
  - If books references the publisher → stop action (or set to NULL)

Referential Actions: Examples…

- Consider the following relational schema:

  Reader (**readerId**, rstName, lastName, address, city, dateOfBirth)
  Book (**isbn**, title, author, numberOfPages, yearOfPublication, publisherName)
  Publisher (**publisherName**, placeOfPublication)
  Categories (**categoryName**, includedIn)
  Copy (**ISBN**, **copyNumber**, shelf, position)
  Loan (**readerId**, **ISBN**, copyNumber, returnDate)
  BookCategory (**ISBN**, **categoryName**)

- Impact of updating a readerId?
  - Modification of the reader ID → update (in table Loan)

Referential Actions: Examples…

- Consider the following relational schema:

  Reader (**readerId**, rstName, lastName, address, city, dateOfBirth)
  Book (**isbn**, title, author, numberOfPages, yearOfPublication, publisherName)
  Publisher (**publisherName**, placeOfPublication)
  Categories (**categoryName**, includedIn)
  Copy (**ISBN**, **copyNumber**, shelf, position)
  Loan (**readerId**, **ISBN**, copyNumber, returnDate)
  BookCategory (**ISBN**, **categoryName**)

- Enforce that only readers who live in Zurich can be inserted.

```
ALTER TABLE Reader
ADD CHECK (city='Zurich')
```

Referential Actions: Examples…

- Consider the following relational schema:

  Reader (**readerId**, rstName, lastName, address, city, dateOfBirth)
  Book (**isbn**, title, author, numberOfPages, yearOfPublication, publisherName)
  Publisher (**publisherName**, placeOfPublication)
  Categories (**categoryName**, includedIn)
  Copy (**ISBN**, **copyNumber**, shelf, position)
  Loan (**readerId**, **ISBN**, copyNumber, returnDate)
  BookCategory (**ISBN**, **categoryName**)

- Enforce that a reader can only borrow up to 20 books [CHECK version]

```
ALTER TABLE Loan
ADD CHECK (
        NOT EXISTS (
                SELECT Count(*) as book_count
                FROM Loan
                GROUP BY readerID
                HAVING book_count > 20  ) )
```

Note: This is not be supported by all databases.

Referential Actions: Examples…

```
CREATE TABLE Beers (
        name CHAR(20) PRIMARY KEY,
        manf CHAR(20) );
```

```
CREATE TABLE Sells (
        bar CHAR(20),
        beer CHAR(20) REFERENCES Beers(name),
        price REAL );
```

```
CREATE TABLE Sells (
        bar CHAR(20),
        beer CHAR(20) CHECK(
              beer IN (SELECT name
                       FROM Beers) ),
        price REAL);
```

=?

Referential Actions: Examples…

```
CREATE TABLE table_x ( id integer NOT NULL, text character varying(255) NOT
NULL, CONSTRAINT table_x_pk PRIMARY KEY (id) );
```

```
CREATE TABLE table_y ( id integer NOT NULL, text character varying(255) NOT
NULL, id_x integer NOT NULL, CONSTRAINT table_y_pk PRIMARY KEY (id) );
```

```
ALTER TABLE table_y ADD CONSTRAINT id_x_fk FOREIGN KEY (id_x) REFERENCES
table_x (id);
```

```
INSERT INTO table_x VALUES (1, 'super x'), (2, 'great x');
INSERT INTO table_y VALUES (1, 'y one', 2), (2, 'y two', 1), (3, 'y three',
1);
```

```
DELETE FROM table_x WHERE id = 1; ERROR…!!!
```

```
ALTER TABLE table_y DROP CONSTRAINT id_x_fk, ADD CONSTRAINT id_x_fk FOREIGN
KEY (id_x) REFERENCES table_x (id) ON DELETE CASCADE;
```

```
DELETE FROM table_x WHERE id = 1;
```

```
TRUNCATE table_x CASCADE;
```

Cascade Scenario

SQL Assertions

- Database-schema constraint.
- Not present in Oracle.
- Checked whenever a mentioned relation changes.
- Syntax:

```
    CREATE ASSERTION < name>
    CHECK(<condition>);
```

Example

- No bar may charge an average of more than $5 for beer.

  Sells(<u>bar</u>, <u>beer</u>, price)

```
CREATE ASSERTION NoRipoffBars
CHECK(NOT EXISTS(
       SELECT bar
       FROM Sells
       GROUP BY bar
       HAVING 5.0 < AVG(price)
       )
);
```

- Checked whenever `Sells` changes.

Example...

- There cannot be more bars than drinkers.

```
Bars(name, addr, license)
Drinkers(name, addr, phone)

CREATE ASSERTION FewBar
CHECK(
    (SELECT COUNT(*) FROM Bars) <=
    (SELECT COUNT(*) FROM Drinkers)
);
```

- Checked whenever `Bars` or `Drinkers` changes.

Triggers: Motivation

- In principle, we must check every assertion after every modification to any relation of the database.
- Assertions are powerful, but the DBMS often can't tell when they need to be checked.
- Attribute- and tuple-based checks are checked at known times, but are not powerful.
- Triggers let the user decide when to check for any condition.

Triggers (Oracle Version)

Often called event-condition-action rules

- *Event* = a class of changes in the DB, *e.g.*, "insertions into `Beers`."
- *Condition* = a test as in a where-clause for whether or not the trigger applies.
- *Action* = one or more SQL statements.
- Differ from checks or SQL assertions in that:
  1. Triggers invoked by the event; the system doesn't have to figure out when a trigger could be violated.
  2. Condition not available in checks.

Trigger: Example 1

Whenever we insert a new tuple into `Sells`, make sure the beer mentioned is also mentioned in `Beers`, and insert it (with a null manufacturer) if not.

```
Sells(bar, beer, price)
```

The event

```
CREATE OR REPLACE TRIGGER BeerTrig
AFTER INSERT ON Sells
FOR EACH ROW
WHEN(new.beer NOT IN
            (SELECT name FROM Beers))
   BEGIN
         INSERT INTO Beers(name)
         VALUES(:new.beer);
   END;
```

The condition

The action

Options

1. Can omit `OR REPLACE`. But if you do, it is an error if a trigger of this name exists.
2. `AFTER` can be `BEFORE`.
3. If the relation is a view, `AFTER` can be `INSTEAD OF`.
   - Useful for allowing "modifications" to a view; you modify the underlying relations instead.
4. `INSERT` can be `DELETE` or `UPDATE OF` <attribute>.
   - Also, several conditions like `INSERT ON Sells` can be connected by `OR`.
5. `FOR EACH ROW` can be omitted, with an important effect: the action is done once for the relation(s) consisting of all changes.

Triggers (PostgreSQL Version)

CREATE TRIGGER trigger_name {**BEFORE** | **AFTER** | **INSTEAD OF**} {event [**OR** ...]}
  ON table_name
  [**FOR** [**EACH**] {**ROW** | **STATEMENT**}]
  EXECUTE PROCEDURE trigger_function

```
CREATE OR REPLACE TRIGGER BeerTrig
AFTER INSERT ON Sells
FOR EACH ROW
EXECUTE PROCEDURE func();
END;
```

```
CREATE FUNCTION func()
RETURNS TRIGGER AS
BEGIN
IF new.beer NOT IN
(SELECT name FROM Beers)) THEN
INSERT INTO Beers(name)
   VALUES(:new.beer)
End;
```

Trigger: Example 2

Maintain a list of all the bars that raise their price for some beer by more than $1.

Sells(<u>bar</u>, <u>beer</u>, price)
RipoffBars(<u>bar</u>)

```
CREATE TRIGGER PriceTrig
AFTER UPDATE OF price ON Sells
FOR EACH ROW
WHEN(new.price > old.price + 1.00)
   BEGIN
        INSERT INTO RipoffBars
        VALUES(:new.bar);
   END;
```

Assignment 5

A sample database is given, refer to the link, try to answer the following set of queries and write down the SQL statement with brief description and exact output.

- Sample Database File

1. List all 4 categories (id and name), sorted on name.
2. List the unique titles (movie), of all movies nominated in 1990, sorted alphabetically.
3. List the unique last names of actors nominated for the Leading Actor category between 2000 and 2002, ordered alphabetically.

Assignment 5…

4. Super-movie(s): List the movie name , and year, for the movie that attracted the highest count of nominations in the history of Oscars. If more than one such movie exists, list them all (movie, year), sorted on movie name.
5. List the most common first name for a person, and the count of people that have this first name. (E.g., return ("Mary", 23), if there are 23 actresses named "Mary", and no other first name is that popular). If there is a tie, give only the alphabetically first one. (Hint: Check the limit keyword.)