Logical Schema Design:
Schema Definition with SQL (DDL)

SQL history and standards

SQL type system

Specifying constraints with SQL

Standard Query Language: Introduction

- No pure relations in DBMS but tables
  - Duplicate data not deleted
  - Tuples – rows, Attributes – columns
- Standard Query Language (SQL)
  - Declarative language for DB
  - Available in all relational DBMS
- Support of two interfaces:
  - Interactive Interface: User friendly Interface (UFI)
  - Application-program Interface: "embedded SQL"

Standard Query Language: History

1974	Prototype "System R" (IBM, San Jose) ▸ First relational DBMS ▸ based on Codd's relational model ▸ Structured English Query Language (SEQUEL)
1975	SEQUEL renamed SQL (pronounced "Sequel" in US)
1986	First standardization attempt based on system R
1989	SQL standard ▸ ANSI SQL-1 , SQL-89 ▸ about 120 pages
1992	SQL2 standard ▸ ANSI SQL-2, SQL-92 ▸ about 600 pages

Standard Query Language: Current standard

- 1999: SQL:1999 standard (ANSI SQL-3)
  - about 2200 pages as full standard

- Various parts (not finished):
  - Framework (SQL/Framework), introduction
  - Foundation (SQL/Foundation), core SQL
  - Call-Level Interface (SQL/CLI), ODBC 3
  - Persistent Stored Modules (SQL/PSM), stored procedures
  - Host Language Bindings (SQL/ Bindings), embedded SQL
  - Temporal data (SQL/Temporal)
  - Management of external data (SQL/MED)
  - Object Language Bindings (SQL/OLB), embedded SQL-Java
  - Multimedia (SQL/MM), full-text and spatial data
  - SQL and XML

Standard Query Language: Standards

- SQL-92 compliance levels:
  (1) Entry SQL: basically SQL-89, essential
  (2) Intermediate SQL,
  (3) Full SQL
  - No implementation of SQL-92 on level 2 or3

- SQL:1999 levels:
  - Core SQL: essential for standard compliance
  - Additional Features, e.g. multimedia

Core SQL:1999

enhanced SQL:1999

- New standards replace old ones
- In DBMS implementations much added / left out

Standard Query Language: Course Information

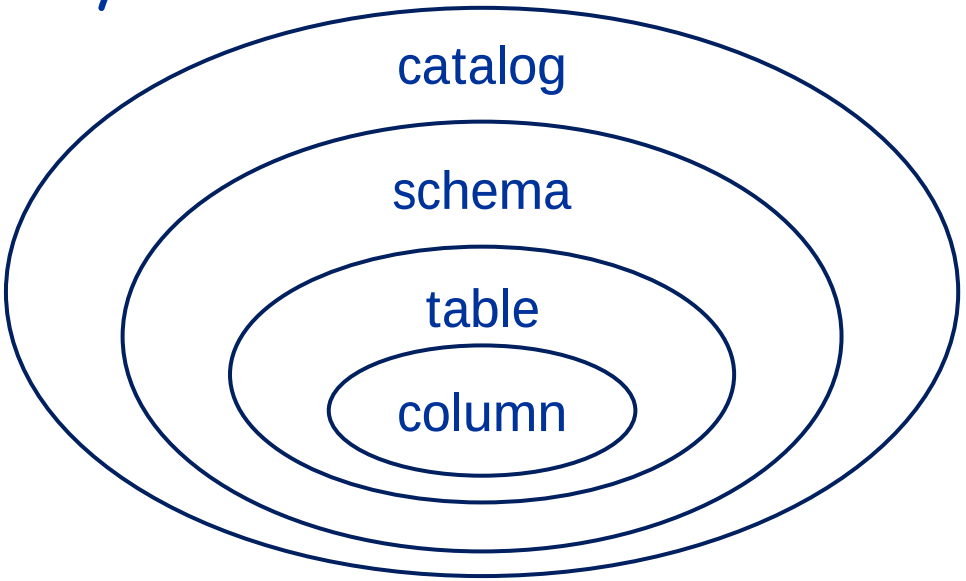
- Within the course:
  - Basic concepts of SQL:1999
  - Oracle10i (commercial)
    Core SQL:1999 compliant + additional features
  - PostgreSQL (open source)
    Core SQL:1999 compliant
  - MySQL (open source)
    traditionally not SQL compliant
    (no subqueries, foreign keys,…)
- Self study of further SQL concepts

Standard Query Language: Components

- Data definition Language (DDL)
  - Definition and change of data structures on all three database levels: Namespaces, relations with attributes, domains, data types, integrity constraints, triggers, functions on the database,views, placement of data, space needed, access structures,...
- Data manipulation language (DML)
  - Create, change, delete data
  - Interactive query formulation
  - Embedding of SQL commands in host language
  - Specification of begin, abort, and end of transaction
- Data Administration language
  - Access rights, authorization

SQL / DDL: SQL Objects

- Examples: Catalog, schema, table, trigger,...
- Descriptor = Object identificator (e.g., name)
- Object hierarchy:



- Catalog:
  - Named group of schemas
  - Created implicitly

SQL / DDL: Schema

- Named group of SQL-objects by particular user

- Creates namespace
  - Unambiguous object names

    <catalog>.<schema>.<table>.<column>

    <catalog>.<schema>.<trigger>

  - Not supported by all systems
  - Always supported: **<table>.<column>**

- Syntax: **CREATE SCHEMA <schemaName>;**

SQL / DDL: Namespaces

- Confusing terminology implemented

- Oracle
  - Database = set of physical storage areas ("tablespaces")
  - Schema name = dbUsername
  - Object names prefixed with <dbUsername>

- PostgreSQL
  - Database = schema
  - Schema name = database name

- MySQL
  - Database = directory in File system where data reside
  - Schema not defined in MySQL

SQL / DDL: Predefined data types

- Basic data types:
  - Numbers
  - Characters, strings
  - Date and time
  - Binary objects

- Type systems of different DBS very different
- Use standard compatible types if possible

SQL / DDL: Predefined data types

Core SQL:1999

- Numeric data types
  - **NUMERIC(p,s)** e.g. 300.00
  - **DECIMAL(p,s)**
  - **INTEGER** (alias: **INT**) e.g. 32767
  - **SMALLINT** small integers
  - **FLOAT(p,s)** e.g. -1E+03
  - **REAL** (for short floats)
  - **DOUBLE** (for long floats)

- Examples:
  - Oracle: `NUMBER(precision, scale)`
  - PostgreSQL: `SMALLINT, INTEGER, BIGINT, REAL, NUMERIC(precision,scale), DECIMAL(precision, scale), MONEY, SERIAL (=autoincrement!)`
  - MySQL: `TINYINT[(M)], SMALLINT[(M)], MEDIUMINT[(M)], INT[(M)], BIGINT[(M)], FLOAT(precision), FLOAT[(M,D)], DOUBLE[(M,D)], DOUBLE PRECISION[(M,D)], REAL[(M,D)], DECIMAL[(M[,D])], NUMERIC[(M[,D])]`

SQL / DDL: Predefined data types

Core SQL:1999

- Some string data types
  - **`CHARACTER(n)`** (fixed length)
  - **`CHARACTER`** (variable lenght)
  - **`CHARACTER VARYING(n)`** (alias: **`VARCHAR(n)`**)
  - **`CLOB`** (Character Large Object, e.g., for large text),
  - **`NCLOB`** (National CLOB)

- Examples:
  - Oracle: `VARCHAR2(size), CHAR(size), CLOB, RAW, LONG RAW`
  - PostgreSQL: `CHARACTER(size), CHAR(size), VARYING(size), VARCHAR(size), TEXT`
  - MySQL: `CHAR(M), VARCHAR(M), TINYTEXT, TEXT, MEDIUMTEXT, LONGTEXT`

SQL / DDL: Predefined data types

Core SQL:1999

- Date data types
  - **DATE** *e.g.* DATE '1993-01-02'
  - **TIME** *e.g.* TIME '13:14:15'
  - **TIMESTAMP** *e.g.* TIMESTAMP '1993-01-02 13:14:15.000001'
  - **INTERVAL FirstUnitofTime [to LastUnitofTime]**
    e.g. INTERVAL '01-01' YEAR TO MONTH

- Examples:
  - Oracle: DATE, INTERVAL DAY TO SECOND, INTERVAL YEAR TO MONTH, TIMESTAMP, TIMESTAMP WITH TIME ZONE, TIMESTAMP WITH LOCAL TIME ZONE
  - PostgreSQL: DATE, TIME, TIME WITH TIMEZONE, TIMESTAMP, INTERVAL
  - MySQL: DATE, DATETIME, TIMESTAMP[(M)], TIME, YEAR[(2|4)]

SQL / DDL: Predefined data types

Core SQL:1999

- Binary data types
  - **BIT[(n)]** e.g. B'01000100'
  - **BLOB[(n)]** e.g. X'49FE'
    (Binary Large Objects, e.g., for multimedia)

- Examples:
  - Oracle: BLOB, BFILE, RAW, LONG RAW, ROWID
  - PostgreSQL: TEXT, BYTEA, or in large object
  - MySQL: TINYBLOB, BLOB, MEDIUMBLOB, LONGBLOB

- Additionally:
  - **BOOLEAN** (true, false or unknown)

SQL / DDL: Domain definition, Type definition

Core SQL:1999

- User-created domains
  - Named sets of values
  - Helps avoiding semantically meaningless operations, e.g., comparing money with length attributes
- Syntax:

```
CREATE DOMAIN <domainName> [AS] <typeDef>;
CREATE TYPE <typeName> as <typeDef> FINAL;
```

- Example:
  - `CREATE DOMAIN Money AS DECIMAL(10,2);`
  - `CREATE TYPE Euro AS DECIMAL(8,2) FINAL;`

  - Oracle:
    - Domain - not supp., Type – implemented differently
  - PostgreSQL:
    - Domain – supp.,Type – implemented differently
  - MySQL:
    - Domain - not supp., Type – not supp.

SQL / DDL: Table definition

- Syntax:

```
CREATE TABLE  <TableName> (
   <attributName><attributeType>[<constraint>]
[, <attributName><attributeType>[<constraint>]]
{, <tableConstraints>});
```

- Example:

```
CREATE TABLE Troll(
  Name   CHAR(10) primary key,
  Height DECIMAL (3,2));
```

- SQL is case-insensitive for restricted words

SQL / DDL: Integrity constraints

Important technique

- Syntax:

  [CONSTRAINT [<name>]]<def>

- Column constraints
  - Example: "must not be NULL", "larger than 1.50"
  - Specified as part of column definition
- Cardinalities
  - Column constraints on keys and foreign keys
- Complex "semantic" constraints ("business rules")
  - Example: "The percentage of movies not older than one year must be 25% or more"
  - More than one row involved, specify after column definitions (table constraint)

SQL / DDL: Integrity constraints

- PRIMARY KEY
  - Only once per table
  - Not necessary, but omission is very bad style
  - Column constraint (single attribute) or table constraint
  - Examples:

```
CREATE TABLE Troll(
  Name   CHAR(10) primary key,
  Height DECIMAL (3,2));

CREATE TABLE Troll(
  Name   CHAR(10) primary key,
  Height DECIMAL (3,2),
  Weight INTEGER,
  CONSTRAINT pk primary key(Name, Height));
```

SQL / DDL: Integrity constraints

- NOT NULL
  - Value must not be NULL
  - Column constraint
  - Example:

```
CREATE TABLE Format (
 name   CHAR(10) primary key,
 charge DECIMAL(3,2) not NULL);
```

- UNIQUE
  - Column contains only unique values
  - Requires NOT NULL
  - Should be used for candidate keys
  - Column constraint, (table constraint)

SQL / DDL: Integrity constraints

- CHECK clause:
  - Defines predicates that must hold for each row

- Examples:

  - Enumeration:

```
CHECK( VALUES IN ('comedy','suspense',
                  'drama','action',‘SciFi'))
```

  - Interval restriction:

```
CHECK(Charge >= 0 AND Charge < 10)
```

SQL / DDL: Integrity constraints

- Column constraint:

```
CREATE TABLE BankAccount(
  accountno NUMBER(10) primary key,
  amount    DECIMAL(9,2) CHECK (amount > 0),
  credit    DECIMAL(7,2));
```

- Multicolumn constraint:

```
CREATE TABLE BankAccount(
  accountno NUMBER(10) primary key,
  amount    DECIMAL(9,2),
  credit    DECIMAL(7,2),
  CONSTRAINT account CHECK (amount+credit>0));
```

SQL / DDL: Referential Integrity

Important concept

- Foreign Key

  Consider relation R with key k and relation S

  $fk \subset S$ is foreign key if for all tuples $s \in S$ holds:

  1. s.fk contains only NULL values or only values ≠ NULL
  2. If s.fk contains no NULL values ∃ tuple $r \in R$: s.fk=r.k

- Referential integrity:

  Foreign key constraint (above) holds

- Referential Integrity in SQL
  - Candidate keys with UNIQUE
  - Primary keys with PRIMARY KEY
  - Foreign keys with REFERENCES

SQL / DDL: Integrity constraints

Important technique

- FOREIGN KEY
  - References keys in other tables
  - Ensures references to existing key values only

```
CREATE TABLE Tape(
  id       INTEGER PRIMARY KEY,
  format   CHAR(5) NOT NULL,
  movie_id INTEGER NOT NULL,
  CONSTRAINT tapeNotEmpty
  FOREIGN KEY (movie_id) REFERENCES Movie(id),
  CONSTRAINT formatCheck
  FOREIGN KEY (format) REFERENCES Format(name)
);
```

SQL / DDL: Integrity constraints

Important technique

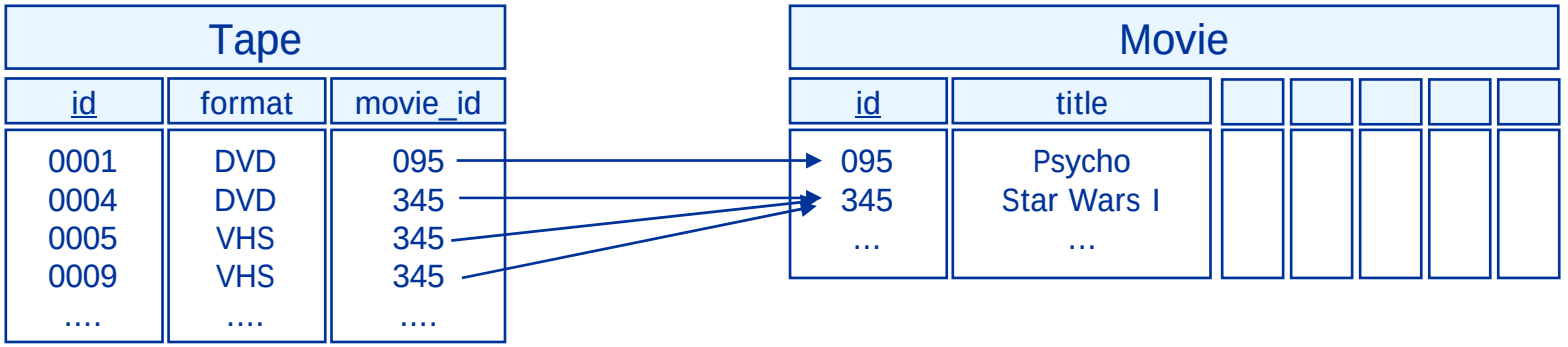
- FOREIGN KEY prevents execution of SQL statements which violate Referential Integrity
- Update and delete may cause violation

- Define actions:
  - On delete cascade: delete all referencing tuples
  - On delete set NULL
  - On delete set default
  - On update cascade: update key in referencing table
  - On update set NULL
  - On update set default

SQL / DDL: Integrity constraints

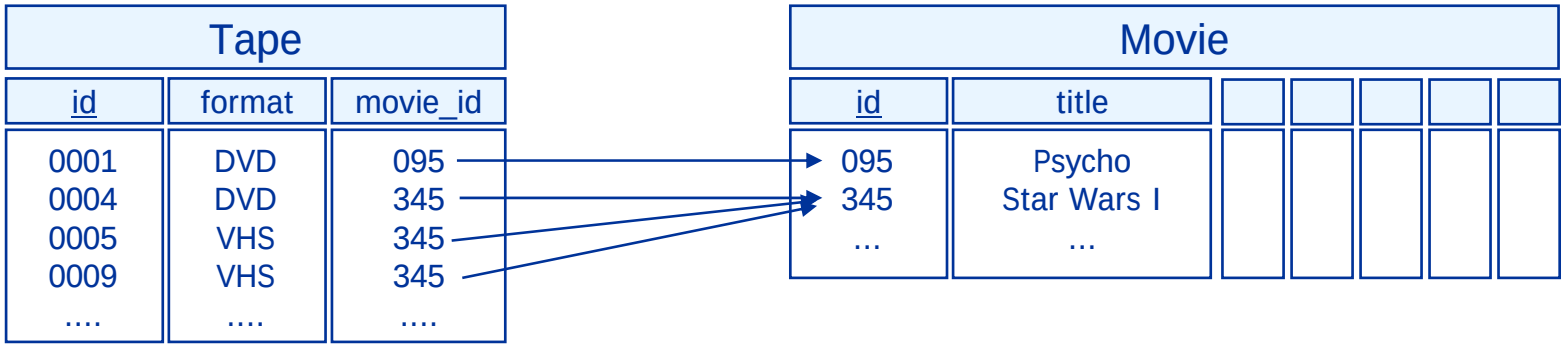
Example:

```
CREATE TABLE Tape(
  id       INTEGER PRIMARY KEY,
  format   CHAR(5) NOT NULL,
  movie_id INTEGER NOT NULL,
  CONSTRAINT tapeNotEmpty
  FOREIGN KEY (movie_id) REFERENCES Movie(id)
  ON DELETE CASCADE,
  CONSTRAINT formatCheck
  FOREIGN KEY (format) REFERENCES Format(name)
  ON DELETE SET NULL);
```

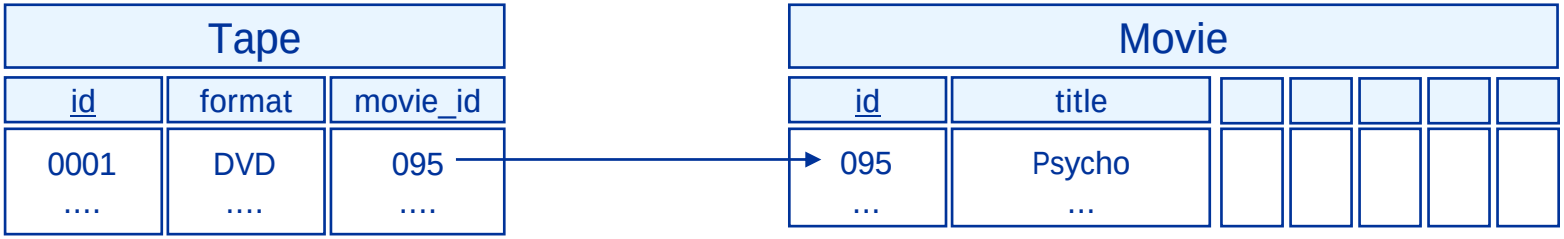


SQL / DDL: Integrity constraints

Example:

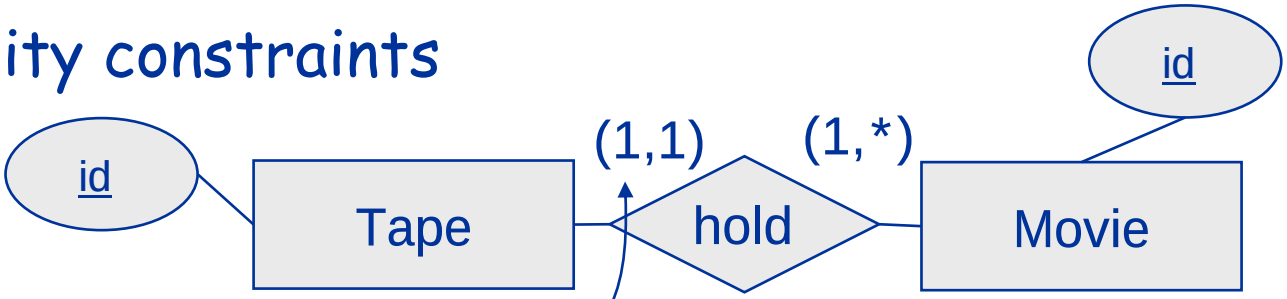



```
Delete from Movie
Where id = 345;
```



SQL / DDL: Integrity constraints

- Cardinality constraints

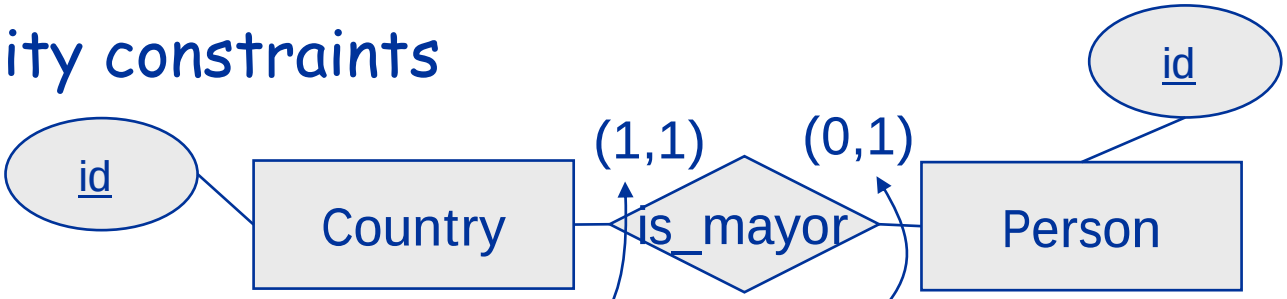


  - NOT NULL ensures min = 1

```
CREATE TABLE Tape(
  id       INTEGER PRIMARY KEY,
  format   CHAR(10) NOT NULL,
  movie_id INTEGER NOT NULL,
  CONSTRAINT tapeNotEmpty
  FOREIGN KEY (movie_id) REFERENCES Movie(id),
  CONSTRAINT formatCheck
  FOREIGN KEY (format) REFERENCES Format(name));
```

SQL / DDL: Integrity constraints

- Cardinality constraints

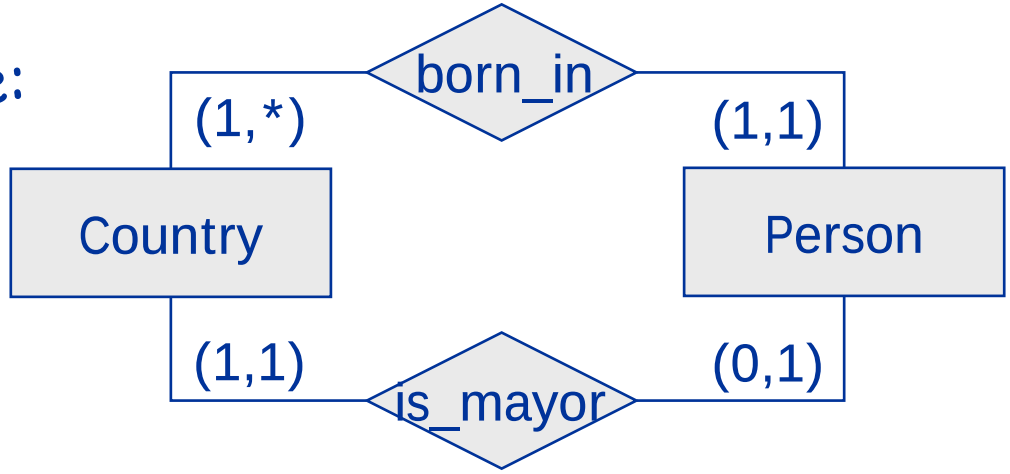


  - NOT NULL ensures min = 1
  - UNIQUE ensures max= 1

```
CREATE TABLE Country(
  id       INTEGER PRIMARY KEY,
  mayor    INTEGER UNIQUE NOT NULL,
CONSTRAINT mayorFK
  FOREIGN KEY (mayor) REFERENCES Person(id));
```

SQL / DDL: Mandatory relationships

- Example:



- How to define "circular" constraints?
  - Specify constraints after table definition

```
ALTER TABLE Person
  ADD (CONSTRAINT birthPlaceReference
  FOREIGN KEY (birthplace)
  REFERENCES country(id));

ALTER TABLE Person
  MODIFY COLUMN( birthplace not null);
```

SQL / DDL: Mandatory relationships

The Chicken-Egg problem

```
CREATE TABLE chicken(cID INT PRIMARY KEY,
                     eID INT);
CREATE TABLE egg(eID INT PRIMARY KEY,
                 cID INT);

    -- Example by J. Widom et al.
ALTER TABLE chicken
    ADD CONSTRAINT chickenREFegg
    FOREIGN KEY (eID) REFERENCES egg(eID);


ALTER TABLE egg
    ADD CONSTRAINT eggREFchicken
    FOREIGN KEY (cID) REFERENCES chicken(cID)  ;
```

What happens if an egg / chicken is inserted?

SQL / DDL: Mandatory relationships

Insertion violates foreign key constraint

```
INSERT INTO chicken VALUES(1, 2);
ORA-02291: integrity constraint
  (chickenREFegg.SYS_C0067174)  violated - parent key
  not found
INSERT INTO egg VALUES(2, 1);
ORA-02291: integrity constraint
  (eggREFchicken.SYS_C0067174)  violated - parent key
  not found
```

Defer constraint checking

```
ALTER TABLE chicken
  ADD CONSTRAINT chickenREFegg
  FOREIGN KEY (eID) REFERENCES egg(eID)
  INITIALLY DEFERRED DEFERRABLE;
```

SQL / DDL: Mandatory relationships

Deferred constraints checked at the end of a transaction.
Transaction: unit of work consisting of one or more operations on the DB

COMMIT closes a transaction

```
INSERT INTO chicken VALUES(1, 2);
-- constraint not checked here
INSERT INTO egg VALUES(2, 1);
COMMIT;  -- but here
```

Variants

```
INITIALLY DEFERRED DEFERRABLE
INITIALLY IMMEDIATE DEFERRABLE
SET CONSTRAINT <name>
               [DEFERED|IMMEDIATE]
```

allow checking at arbitrary times

SQL / DDL: Mandatory relationships

- Trigger
  - Specify trigger on table Movie:

```
CREATE TRIGGER alwaysTape
   AFTER INSERT ON Movie
   REFERENCING NEW ROW AS m
   FOR EACH ROW WHEN
                  NOT EXISTS (
                              SELECT *
                              From Tape
                              WHERE movie_id=m.id)
      <do something>;
```

  - Very flexible and expressive
  - Later more…

SQL / DDL: Alter table definitions

- Syntax:

```
ALTER TABLE<tableName><redefinition>;
```

- Add an attribute

```
ALTER TABLE Tape
  ADD COLUMN (AquiredBy char(20));
```

- Change an attribute
  - Attribute names cannot be changed

```
ALTER TABLE Tape
  MODIFY (AquiredBy char(30));
```

- Add constraints,...

SQL / DDL: More statements

- Delete table
  - Delete table only if not referenced

```
DROP TABLE <tableName> restrict
```

  - Delete table and reference

```
DROP TABLE <tableName> cascade;
```

    Oracle: `DROP TABLE <tableName> cascade constraints;`

- Attributes can have default values

```
<attributeName> <attributeType> DEFAULT <value>
```

SQL / DDL: Example

```
CREATE TABLE Movie (
  id         INTEGER PRIMARY KEY,
  title      VARCHAR(60) NOT NULL,
  category   CHAR(10),
  year       DATE,
  director   VARCHAR(30),
  pricePDay  DECIMAL(4,2),
  length     INTEGER,
  CONSTRAINT plausible_year
   CHECK (year > TO_DATE('01.01.1900','DD.MM.YYYY')),
  CONSTRAINT allowedPrice
   CHECK ( (pricePDay >= 0) AND (pricePDay < 100.0))
 );
```

SQL / DDL: Example

```
CREATE TABLE Format(
  Name   CHAR(5) primary key,
  Charge DECIMAL (3,2)
);

CREATE TABLE Tape(
  id       INTEGER PRIMARY KEY,
  format   CHAR(5) NOT NULL,
  movie_id INTEGER NOT NULL,
  CONSTRAINT tapeNotEmpty
  FOREIGN KEY (movie_id) REFERENCES Movie(id)
  ON DELETE CASCADE,
  CONSTRAINT formatCheck
  FOREIGN KEY (format) REFERENCES Format(name)
  ON DELETE SET NULL
);
```

SQL / DDL: Example

```
CREATE TABLE Actor (
 stage_name VARCHAR(30)NOT NULL UNIQUE,
 real_name  VARCHAR(30),
 birthday DATE
);


CREATE TABLE Play (
  movie_id    INTEGER,
  actor_name  VARCHAR(30),
 CONSTRAINT pkStarr
   PRIMARY KEY (movie_id, actor_name),
 CONSTRAINT foreignKeyMovieID
   FOREIGN KEY (movie_id)
   REFERENCES Movie (id),
 CONSTRAINT foreignKeyStagename
   FOREIGN KEY (actor_name)
   REFERENCES Actor(stage_name)
);
```

SQL / DDL: Example

```
CREATE TABLE Customer (
 mem_no      INTEGER PRIMARY KEY,
 last_name   VARCHAR (30) NOT NULL,
 first_name  VARCHAR(20),
 address     VARCHAR (60),
 telephone   VARCHAR (15));


CREATE TABLE Rental(
 tape_id     INTEGER,
 mem_no      INTEGER,
 from_date   DATE NOT NULL,
 until_date  DATE,
 PRIMARY KEY (tape_id, mem_no, from_date),
 CONSTRAINT fk_Tape
    FOREIGN KEY (tape_id) REFERENCES Tape(id),
 CONSTRAINT fk_Customer
    FOREIGN KEY (mem_no)
    REFERENCES Customer(mem_no));
```

SQL / DDL: Implementations

- Oracle
  - PRIMARY KEY, NOT NULL, UNIQUE, FOREIGN KEY, REFERENCES, CHECK supported
  - Unique artificial key values: use sequence-object
- PostgresSQL
  - PRIMARY KEY, NOT NULL, UNIQUE, FOREIGN KEY, REFERENCES, CHECK supported
  - Unique artificial key values: serial data type and sequences
- MySQL:
  - PRIMARY KEY, NOT NULL, UNIQUE supported
  - FOREIGN KEY, REFERENCES, CHECK accepted (for compatibility) but not supported
  - Unique artificial key values: **`AUTO_INCREMENT`**
  - Many changes in new versions (!)

SQL / DDL: Metadata management

- Metadata: SQL object definitions
- Stored in system data structures ("the catalogue")
- Mostly accessed as tables
  - All user-tables:

```
SELECT table_name
FROM user_tables;
```

  - All constraints:

```
SELECT constraint_name,search_condition
FROM   user_constraints
WHERE  table_name = 'MOVIE';
```

```
CONSTRAINT_NAME  C SEARCH_CONDITION
-----------------------------------------------------------------
SYS_C002723      C "TITLE" IS NOT NULL
PLAUSIBLE_YEAR   C year > TO_DATE('01.01.1900','DD.MM.YYYY')
ALLOWEDPRICE     C  (pricePDay >= 0) AND (pricePDay < 100.0)
SYS_C002726      P
```

SQL / DDL: Summary

- Standard Query Language (SQL)
  - Data definition language (DDL)
  - Data manipulation language (DML)
  - In almost all current DBMS
  - All SQL implementations differ from standard

- Important terms and concepts:
  - Data types
  - Create, change, delete tables
  - Referential integrity
  - Integrity constraints