# Lic. Informatique, Univ Lille 1, 2010-11

- **Introduction aux bases de données relationnelles**
  - 6ème séance: **SQL, niveau intermédiaire**
- **Enseignante**: C. Kuttler
- **Biblio**: chapitre 4 de *Database Systems Concepts* de Silberschatz et al, McGraw-Hill (6ème edition, 2010)
- Ces transparents sont une adaptation de ceux disponibles sur le site du livre: **www.db-book.com**

# Chapter 4: Intermediate SQL

- Null values in queries (Chap 3)
- Join Expressions
- Table creation (already seen on 10/09 Chap 3)
- Table modification and updates (Chap 3 )
- Integrity Constraints
- SQL Data Types and Schemas
- Authorization
- Views
- Transactions

# Null Values: the Tricky Details

- It is possible for tuples to have a null value, denoted by *null*, for some of their attributes
- *null* signifies **an unknown value or that a value does not exist.**
- The result of any arithmetic expression involving *null* is *null*
  - Example: 5 + *null* returns null
- The predicate **is null** can be used to check for null values.
  - Example: Find all instructors whose salary is null.

```
select name
from instructor
where salary is null
```

# Three Valued Logic: true, false, unkown

- Result of **where** clause predicate is treated as *false* if it evaluates to *unknown*
- Any comparison with *null* returns *unknown*
  - Example*: 5 < null* *or* *null <> null* *or* *null = null*
- Three-valued logic using the truth value *unknown*:
  - OR: (*unknown* **or** *true*) = *true*,
    (*unknown* **or** *false*) = *unknown*
    (*unknown* **or** *unknown*) = *unknown*
  - AND: *(true* **and** *unknown)* = *unknown,*
    *(false* **and** *unknown)* = *false,*
    *(unknown* **and** *unknown)* = *unknown*
  - NOT*:* *(***not** *unknown)* = *unknown*
  - "*P* **is unknown**" evaluates to true if predicate *P* evaluates to *unknown*

# Null Values and Aggregates

# Null Values and Aggregates

- Total all salaries

  **select sum** (*salary* )
  **from** *instructor*

# Null Values and Aggregates

- Total all salaries

  **select sum** (*salary* )
  **from** *instructor*

  - Above statement ignores null amounts

# Null Values and Aggregates

- Total all salaries

  **select sum** (*salary* )
  **from** *instructor*

  - Above statement ignores null amounts
  - Result is *null* if there is no non-null amount

# Null Values and Aggregates

- Total all salaries

  **select sum** (*salary* )
  **from** *instructor*

  - Above statement ignores null amounts
  - Result is *null* if there is no non-null amount

- All aggregate operations except **count(*)** ignore tuples with null values on the aggregated attributes

# Null Values and Aggregates

- Total all salaries

  **select sum** (*salary* )
  **from** *instructor*

  - Above statement ignores null amounts
  - Result is *null* if there is no non-null amount
- All aggregate operations except **count(*)** ignore tuples with null values on the aggregated attributes
- What if collection has only null values?

# Null Values and Aggregates

- Total all salaries

  **select sum** (*salary* )
  **from** *instructor*

  - Above statement ignores null amounts
  - Result is *null* if there is no non-null amount
- All aggregate operations except **count(*)** ignore tuples with null values on the aggregated attributes
- What if collection has only null values?
  - count returns 0

# Null Values and Aggregates

- Total all salaries

  **select sum** (*salary* )
  **from** *instructor*

  - Above statement ignores null amounts
  - Result is *null* if there is no non-null amount
- All aggregate operations except **count(*)** ignore tuples with null values on the aggregated attributes
- What if collection has only null values?
  - count returns 0
  - all other aggregates return null

# Chapter 4: Intermediate SQL

- Null values in queries (Chap 3)
- Join Expressions
- Table creation (already seen on 10/09 Chap 3)
- Table modification and updates (Chap 3 )
- Integrity Constraints
- SQL Data Types and Schemas
- Authorization
- Views
- Transactions

# Joined Relations

- **Join operations** take two relations and return as a result another relation.
- A join operation is a Cartesian product which requires that tuples in the two relations match (under some condition). It also specifies the attributes that are present in the result of the join.
- **Join condition** – defines which attributes in the two relations match
- **Join type** – defines how tuples in each relation that do not match any tuple in the other relation (based on the join condition) are treated.

| *Join types* |
| --- |
| **inner join** |
| **left outer join** |
| **right outer join** |
| **full outer join** |

| *Join Conditions* |
| --- |
| **natural** |
| **on** <predicate> |
| **using** $(A_1, A_1, \ldots, A_n)$ |

# Natural Join

- Natural join matches tuples with the same values for all common attributes, and retains only one copy of each common column
- **select** *
  **from** *instructor* **natural join** *teaches*;

| *ID* | *name* | *dept_name* | *salary* | *course_id* | *sec_id* | *semester* | *year* |
|---|---|---|---|---|---|---|---|
| 10101 | Srinivasan | Comp. Sci. | 65000 | CS-101 | 1 | Fall | 2009 |
| 10101 | Srinivasan | Comp. Sci. | 65000 | CS-315 | 1 | Spring | 2010 |
| 10101 | Srinivasan | Comp. Sci. | 65000 | CS-347 | 1 | Fall | 2009 |
| 12121 | Wu | Finance | 90000 | FIN-201 | 1 | Spring | 2010 |
| 15151 | Mozart | Music | 40000 | MU-199 | 1 | Spring | 2010 |
| 22222 | Einstein | Physics | 95000 | PHY-101 | 1 | Fall | 2009 |
| 32343 | El Said | History | 60000 | HIS-351 | 1 | Spring | 2010 |
| 45565 | Katz | Comp. Sci. | 75000 | CS-101 | 1 | Spring | 2010 |
| 45565 | Katz | Comp. Sci. | 75000 | CS-319 | 1 | Spring | 2010 |
| 76766 | Crick | Biology | 72000 | BIO-101 | 1 | Summer | 2009 |
| 76766 | Crick | Biology | 72000 | BIO-301 | 1 | Summer | 2010 |

# Join operations – More examples

- Relation *course*

| *course_id* | *title* | *dept_name* | *credits* |
|---|---|---|---|
| BIO-301 | Genetics | Biology | 4 |
| CS-190 | Game Design | Comp. Sci. | 4 |
| CS-315 | Robotics | Comp. Sci. | 3 |

- Relation *prereq*

| *course_id* | *prereq_id* |
|---|---|
| BIO-301 | BIO-101 |
| CS-190 | CS-101 |
| CS-347 | CS-101 |

- Note: no prerequisite for CS-315, and no course Information for CS-437.

# Inner join

- *course* **inner join** *prereq* **on** *course.course_id = prereq.course_id*

**Correction: prereq_id**

| *course_id* | *title* | *dept_name* | *credits* | *prere_id* | *course_id* |
|---|---|---|---|---|---|
| BIO-301 | Genetics | Biology | 4 | BIO-101 | BIO-301 |
| CS-190 | Game Design | Comp. Sci. | 4 | CS-101 | CS-190 |

- **Equivalent to:**

**select *  from** *course, prereq*  **where** *course.course_id =prereq.course_id*

*course* ***join*** *prereq* ***on*** *course.course_id=prereq.course_id*

# Outer Join

- Avoids loss of information.
- Computes the join
  - then adds tuples from one relation for which there is no match in the other relation to the result of the join.
  - Fills the empty attributes in the result with *null* values.

# Left Outer Join

- *course* **natural left outer join** *prereq*
- *CS-315 doensn't have any prerequisite!*
  - *Corresponding line padded with null value*

**Correction: prereq_id**

| *course_id* | *title* | *dept_name* | *credits* | *prere_id* |
|---|---|---|---|---|
| BIO-301 | Genetics | Biology | 4 | BIO-101 |
| CS-190 | Game Design | Comp. Sci. | 4 | CS-101 |
| CS-315 | Robotics | Comp. Sci. | 3 | *null* |

# Right Outer Join

- *course* **natural right outer join** *prereq*
- course CS-347, occuring in the prerequisite table, isn't listed in the course table! Corresponding attributes are padded with null values in result.

| *course_id* | *title* | *dept_name* | *credits* | *prere_id* |
|---|---|---|---|---|
| BIO-301 | Genetics | Biology | 4 | BIO-101 |
| CS-190 | Game Design | Comp. Sci. | 4 | CS-101 |
| CS-347 | *null* | *null* | *null* | CS-101 |

# Full Outer Join

- *course* **natural full outer join** *prereq*

| *course_id* | *title* | *dept_name* | *credits* | *prere_id* |
|---|---|---|---|---|
| BIO-301 | Genetics | Biology | 4 | BIO-101 |
| CS-190 | Game Design | Comp. Sci. | 4 | CS-101 |
| CS-315 | Robotics | Comp. Sci. | 3 | *null* |
| CS-347 | *null* | *null* | *null* | CS-101 |

# Questions

- How to use outer joins on to detect anomalies on INFOTOUR?
  - Find the identifier of a director who's not listed in the PERSONNES table.
  - Check for other possible violations of what is shown as foreign key constraints in INFOTOUR's diagram.
- How can we obtain the same information, about tuples of one table for which there is no matching data in another table, without outer joins?
- Other useful queries on the university database with outer joins?

# Chapter 4: Intermediate SQL

- Null values in queries (Chap 3)
- Join Expressions
- Table creation (already seen on 10/09)
- Table modification and updates (Chap 3 )
- Integrity Constraints
- SQL Data Types and Schemas
- Authorization
- Views
- Transactions

# Create Table Construct

- An SQL relation is defined using the **create table** command:

  **create table** *r* ($A_1$ $D_1$, $A_2$ $D_2$, ..., $A_n$ $D_n$,
  (integrity-constraint$_1$),
  ...,
  (integrity-constraint$_k$))

  - *r* is the name of the relation
  - each $A_i$ is an *attribute name* in the schema of relation *r*
  - $D_i$ is the data type of values in the domain of attribute $A_i$

- Example:

  **create table** *instructor* (
  *ID* **char**(5),
  *name* **varchar**(20) **not null,**
  *dept_name* **varchar**(20),
  *salary* **numeric**(8,2))

# Integrity Constraints in Create Table

- **not null**
- **default value**
- **primary key** ($A_1$, ..., $A_n$)
- **foreign key** ($A_1$, ..., $A_n$) **references** $r$

Example: Declare *branch_name* as the primary key for *branch*

```
create table instructor (
      ID              char(5),
      name            varchar(20) not null,
      dept_name       varchar(20),
      salary          numeric(8,2),
    primary key (ID),
    foreign key (dept_name) references department))
```

**primary key** declaration on an attribute automatically ensures **not null**

# And a Few More Relation Definitions

- **create table** *student* (
  *ID* **varchar**(5) **primary key**,
  *name* **varchar**(20) not null,
  *dept_name* **varchar**(20),
  *tot_cred* **numeric**(3,0),
  **foreign key** *(dept_name)* **references** *department)* );

- **create table** *takes* (
  *ID* **varchar**(5) **primary key**,
  *course_id* **varchar**(8),
  *sec_id* **varchar**(8),
  *semester* **varchar**(6),
  *year* **numeric**(4,0),
  *grade* **varchar**(2),
  **foreign key** (*ID*) **references** *student,*
  **foreign key** (*course_id, sec_id, semester, year*) **references** *section* );

# Chapter 4: Intermediate SQL

- Null values in queries (Chap 3)
- Join Expressions
- Table creation (already seen on 10/09 Chap 3)
- Table modification and updates
- Integrity Constraints
- SQL Data Types and Schemas
- Authorization
- Views
- Transactions

# Drop and Alter Table Constructs

- **drop table**
- **adding and removing attributes to existing table**
  - **alter table** *r* **add** *A D*
    - where *A* is the name of the attribute to be added to relation *r* and *D* is the domain of *A.*
    - All tuples in the relation are assigned *null* as the value for the new attribute.
  - **alter table** *r* **drop** *A*
    - where *A* is the name of an attribute of relation *r*
    - Dropping of attributes not supported by many databases.
- **adding constraints to existing table**
  - **alter table** table-name **add** constraint
  - System first ensures that the relation satisfies the constraint. If it does, the constraint is added to the relation. If not, the

# Modification of the Database – Insertion

- Add a new tuple to *course*

  **insert into** *course*
  **values** ('CS-437', 'Database Systems', 'Comp. Sci.', 4);

- or equivalently

  **insert into** *course* (*course_id*, *title*, *dept_name*, *credits*)
  **values** ('CS-437', 'Database Systems', 'Comp. Sci.', 4);

- Add a new tuple to *student* with *tot_creds* set to null

  **insert into** *student*
  **values** ('3003', 'Green', 'Finance', *null*);

# Modification of the Database – Insertion 2

- Insertion of tuples based on an existing table
- Add all instructors to the *student* relation with tot_creds set to 0

```
insert into student
    select ID, name, dept_name, 0
    from   instructor
```

- The **select from where** statement is evaluated fully before any of its results are inserted into the relation
- (otherwise queries like
  **insert into** *table*1 **select** * **from** *table*1
  would cause problems)

# Modification of the Database – Updates

- Increase salaries of instructors whose salary is over $100,000 by 3%, and all others receive a 5% raise
  - Write two **update** statements:

```
update instructor
    set salary = salary * 1.03
    where salary > 100000;
update instructor
    set salary = salary * 1.05
    where salary <= 100000;
```

  - The order is important
  - Someone earning just under 10000$ may obtain two increases!
  - Can be done better using the **case** statement (next slide)

# Case Statement for Conditional Updates

- Same query as before but with case statement

```
update instructor
    set salary = case
                    when salary <= 100000 then salary * 1.05
                    else salary * 1.03
                 end
```

# Updates with Scalar Subqueries

- **Scalar subquery: returns a value (and not: a table)**
- Recompute and update tot_creds value for all students

```
update student S
   set tot_cred = ( select sum(credits)
                    from takes natural join course
                    where S.ID= takes.ID and
                          takes.grade <> 'F' and
                          takes.grade is not null);
```

- Sets *tot_creds* to null for students who have not taken any course
- Instead of **sum**(*credits*), use:

```
case
   when sum(credits) is not null then sum(credits)
   else 0
end
```

# Modification of the Database – Deletion

- Delete all instructors

  **delete from** *instructor*

- Delete all instructors from the Finance department

  **delete from** *instructor*
  **where** *dept_name*= 'Finance';

- Delete all tuples in the *instructor* relation, for those instructors associated with a department located in the Watson building.

  **delete from** *instructor*
  **where** *dept name* **in** (**select** *dept name*
  **from** *department*
  **where** *building* = 'Watson');

# Example Query

- Delete all instructors whose salary is less than the average salary of instructors

# Example Query

- Delete all instructors whose salary is less than the average salary of instructors

**delete from** *instructor*
**where** *salary*< (**select avg** (*salary*) **from** *instructor*);

# Example Query

- Delete all instructors whose salary is less than the average salary of instructors

**delete from** *instructor*
**where** *salary*< (**select avg** (*salary*) **from** *instructor*);

  - Problem: as we delete tuples from deposit, the average salary changes
  - Solution used in SQL:
    1. First, compute **avg** salary and find all tuples to delete
    2. Next, delete all tuples found above (without recomputing **avg** or retesting the tuples)

# Chapter 4: Intermediate SQL

- Null values in queries (Chap 3)
- Join Expressions
- Table creation (already seen on 10/09 Chap 3)
- Table modification and updates (Chap 3 )
- **Integrity Constraints**
- SQL Data Types and Schemas
- Authorization
- Views
- Transactions

# Integrity Constraints

- Integrity constraints guard against accidental damage to the database, by ensuring that authorized changes to the database do not result in a loss of data consistency.
  - A checking account must have a balance greater than $10,000.00.
  - A salary of a bank employee must be at least $4.00 an hour.
  - A customer must have a (non-null) phone number.

# Constraints on a Single Relation

- **not null**
- **default value**
- **primary key**
- **unique**
- **check** (P), where P is a predicate

# Not Null and Unique Constraints

- **not null**
  - Declare *name* and *budget* to be **not null**

    *name* **varchar**(20) **not null**
    *budget* **numeric**(12,2) **not null**

- **unique** ( $A_1$, $A_2$, …, $A_m$)
  - The unique specification states that the attributes *A*1, *A*2, … *A*m form a candidate key.
  - Candidate keys are permitted to be null (in contrast to primary keys).

# The check clause

- **check** (P)

  where P is a predicate

  Example: ensure that semester is one of fall, winter, spring or summer:

```
create table section (
    course_id varchar (8),
    sec_id varchar (8),
    semester varchar (6),
    year numeric (4,0),
    building varchar (15),
    room_number varchar (7),
    time slot id varchar (4),
    primary key (course_id, sec_id, semester, year),
    check (semester in ('Fall', 'Winter', 'Spring', 'Summer'))
);
```

# Referential Integrity

- Ensures that a value that appears in one relation for a given set of attributes also appears for a certain set of attributes in another relation.
  - Example: If "Biology" is a department name appearing in one of the tuples in the *instructor* relation, then there exists a tuple in the *department* relation for "Biology".
- Let A be a set of attributes. Let R and S be two relations that contain attributes A and where A is the primary key of S. A is said to be a **foreign key** of R if for any values of A appearing in R these values also appear in S.

# Cascading Actions in Referential Integrity

- **create table** *course (*
  *course_id* **char**(5) **primary key**,
  *title* **varchar**(20),
  *dept_name* **varchar**(20) **references** *department*
  *)*
- **create table** *course* (
  …
  *dept_name* **varchar**(20),
  **foreign key** (*dept_name*) **references** *department*
  **on delete cascade**
  **on update cascade**,
  . . .
  )
- alternative actions to cascade: **set null**, **set default**

# Integrity Constraint Violation During Transactions

- E.g.,

  **create table** *person* (
  *ID* **char**(10),
  *name* **char**(40),
  *mother* **char**(10),
  *father* **char**(10),
  **primary key** *ID,*
  **foreign key** *father* **references** *person,*
  **foreign key** *mother* **references** *person*)

- How to insert a tuple?
- What if *mother* or *father* is declared not null?
  - **constraint** *father_ref* **foreign key** *father* **references** *person,*
    **constraint** *mother_ref* **foreign key** *mother* **references** *person*)
  - **set constraints** *father_ref, mother_ref* **deferred**

# Complex Check Clauses

- **check** (*time_slot_id* **in** (**select** *time_slot_id* **from** *time_slot*))
  - why not use a foreign key here?
- Every section has at least one instructor teaching the section.
  - how to write this?
- Unfortunately: subquery in check clause not supported by pretty much any database
  - Alternative: triggers (later)
- **create assertion** <assertion-name> **check** <predicate>;
  - Also not supported by anyone

# Chapter 4: Intermediate SQL

- Discussed today:
- Null values in queries (Chap 3)
- Join Expressions
- Table creation (already seen on 10/09 Chap 3)
- Table modification and updates (Chap 3 )
- **Integrity Constraints**