Relational Model

Relational Database: Definitions

- *Relational database:* a set of *relations*

- a relation is a *set* of rows or *tuples* (i.e., all rows are distinct).

Relation: made up of 2 parts:

- *Instance* (set of records): a *table*, with rows and columns.
  #Rows = *cardinality*, #fields = *degree / arity.*

- *Schema* : specifies name of relation, plus name and domain (type) of each field (column).
  - E.G. Students(*sid*: string, *name*: string, *login*: string, *age*: integer, *gpa*: real).

Example Instance of Students Relation

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@eecs	18	3.2
53650	Smith	smith@math	19	3.8

- Cardinality = 3, degree = 5, all rows distinct

Relation Schema

- Formally, given domains $D_1, D_2, \ldots D_n$ a **relation** r is a subset of

  $D_1 \times D_2 \times \ldots \times D_n$

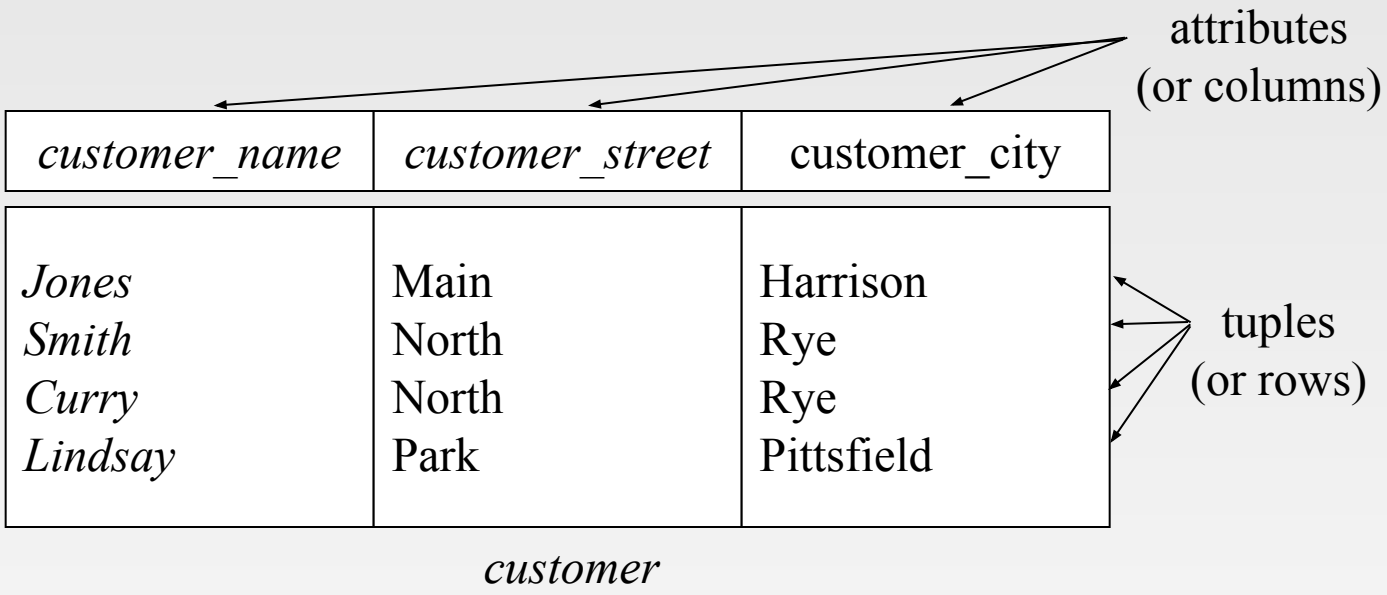
  Thus, a relation is a set of n-tuples $(a_1, a_2, \ldots, a_n)$ where each $a_i \in D_i$
- Schema of a relation consists of
  - attribute definitions
    - name
    - type/domain
  - integrity constraints

Attribute Types

- Each attribute of a relation has a name
- The set of allowed values for each attribute is called the **domain** of the attribute
- Attribute values are (normally) required to be **atomic**; that is, indivisible
  - E.g. the value of an attribute can be an account number, but cannot be a set of account numbers
- Domain is said to be atomic if all its members are atomic
- The special value *null* is a member of every domain
- The null value causes complications in the definition of many operations
  - We shall ignore the effect of null values in our main presentation and consider their effect later

Relation Instance

- The current values (*relation instance*) of a relation are specified by a table
- An element *t* of *r* is a *tuple*, represented by a *row* in a table
- Order of tuples is irrelevant (tuples may be stored in an arbitrary order)



Database

- A database consists of multiple relations
- Information about an enterprise is broken up into parts, with each relation storing one part of the information
- E.g.

account : information about accounts
depositor : which customer owns which account
customer : information about customers

The *customer* Relation

customer_name	*customer_street*	*customer_city*
Adams	Spring	Pittsfield
Brooks	Senator	Brooklyn
Curry	North	Rye
Glenn	Sand Hill	Woodside
Green	Walnut	Stamford
Hayes	Main	Harrison
Johnson	Alma	Palo Alto
Jones	Main	Harrison
Lindsay	Park	Pittsfield
Smith	North	Rye
Turner	Putnam	Stamford
Williams	Nassau	Princeton

The *depositor* Relation

customer_name	*account_number*
Hayes	A-102
Johnson	A-101
Johnson	A-201
Jones	A-217
Lindsay	A-222
Smith	A-215
Turner	A-305

Why Split Information Across Relations?

- Storing all information as a single relation such as
  bank(*account_number, balance, customer_name*, ..)
  results in
  - repetition of information
    - e.g.,if two customers own an account
  - the need for null values
    - e.g., to represent a customer without an account
- Normalization theory deals with how to design relational schemas

Query Languages

- Language in which user requests information from the database.
- Categories of languages
  - Procedural
  - Non-procedural, or declarative
- “Pure” languages:
  - Relational algebra
  - Tuple relational calculus
  - Domain relational calculus
- Pure languages form underlying basis of query languages that people use.

Relational Query Languages

- A major strength of the relational model: supports simple, powerful *querying* of data.

- Queries can be written intuitively, and the DBMS is responsible for efficient evaluation.
  - The key: precise semantics for relational queries.
  - Allows the optimizer to extensively re-order operations, and still ensure that the answer does not change.

The SQL Query Language

- To find all 18 year old students, we can write:

```
SELECT *
FROM Students S
WHERE S.age=18
```

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@ee	18	3.2

- To find just names and logins, replace the first line:

```
SELECT S.name, S.login
```

Querying Multiple Relations

- What does the following query compute?

```
SELECT  S.name, E.cid
FROM  Students S, Enrolled E
WHERE  S.sid=E.sid AND E.grade="A"
```

Given the following instances of Enrolled and Students:

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@eecs	18	3.2
53650	Smith	smith@math	19	3.8

sid	cid	grade
53831	Carnatic101	C
53831	Reggae203	B
53650	Topology112	A
53666	History105	B

we get:

S.name	E.cid
Smith	Topology112

Creating Relations in SQL

- Creates the Students relation. Observe that the type (domain) of each field is specified, and enforced by the DBMS whenever tuples are added or modified.

```
CREATE TABLE Students
(sid CHAR(20),
 name CHAR(20),
 login CHAR(10),
 age INTEGER,
 gpa REAL)
```

- As another example, the Enrolled table holds information about courses that students take.

```
CREATE TABLE Enrolled
(sid CHAR(20),
 cid CHAR(20),
 grade CHAR(2))
```

Adding and Deleting Tuples

- Can insert a single tuple using:

```
INSERT INTO  Students (sid, name, login, age, gpa)
VALUES  (53688, 'Smith', 'smith@ee', 18, 3.2)
```

- Can delete all tuples satisfying some condition (e.g., name = Smith):

```
DELETE
FROM Students S
WHERE S.name = 'Smith'
```

- *Powerful variants of these commands are available; more later!*

Updating Tuples

- Can modify columns in a tuple using:

```
UPDATE  Students S
SET   S.age = S.age + 1, S.gpa = S. gpa - 1,
Where S.sid = 53688
```

- If column is used in determining how rows are updated its old value is used

```
UPDATE  Students S
SET   S.gpa = S. gpa – 0.1,
WHERE S.gpa > 3.3
```

- *Powerful variants of these commands are available; more later!*

Destroying and Altering Relations

```
DROP TABLE Students
```

- Destroys the relation Students. The schema information *and* the tuples are deleted.

```
ALTER TABLE Students
    ADD COLUMN firstYear: integer
```

- The schema of Students is altered by adding a new field; every tuple in the current instance is extended with a *null* value in the new field.

Integrity Constraints (ICs)

- IC: condition that must be true for *any* instance of the database; e.g., *domain constraints.*
  - ICs are specified when schema is defined.
  - ICs are checked when relations are modified.
- A *legal* instance of a relation is one that satisfies all specified ICs.
  - DBMS should not allow illegal instances.
- If the DBMS checks ICs, stored data is more faithful to real-world meaning.
  - Avoids data entry errors, too!

Specifying Constraints in Data Models

- **ER model**
  - **domain and key constraints over entities**
  - **participation and cardinality constraints over relationships**
- **Relational Model**
  - **domain constraints, entity identity, key constraint, functional dependencies -- generalization of key constraints, referential integrity, inclusion dependencies -- generalization of referential integrity.**

- Current database systems support such general constraints in the form of *table constraints* and *assertions*.
- Table constraints are associated with a single table and checked whenever that table is modified.
- In contrast, assertions involve several tables and are checked whenever any of these tables is modified.

Types of constraints

- NOT NULL
- UNIQUE
- DEFAULT
- CHECK
- Key Constraints – PRIMARY KEY, FOREIGN KEY
- Domain constraints
- Mapping constraints

NOT NULL:

NOT NULL constraint makes sure that a column does not hold NULL value. When we don't provide value for a particular column while inserting a record into a table, it takes NULL value by default. By specifying NOT NULL constraint, we can be sure that a particular column(s) cannot have NULL values.

Example:

```
CREATE TABLE STUDENT( ROLL_NO INT NOT NULL,
STU_NAME VARCHAR (35) NOT NULL,
STU_AGE INT NOT NULL,
STU_ADDRESS VARCHAR (235),
PRIMARY KEY (ROLL_NO) );
```

UNIQUE:

UNIQUE Constraint enforces a column or set of columns to have unique values. If a column has a unique constraint, it means that particular column cannot have duplicate values in a table.

```
CREATE TABLE STUDENT( ROLL_NO INT NOT NULL,
STU_NAME VARCHAR (35) NOT NULL UNIQUE,
 STU_AGE INT NOT NULL,
STU_ADDRESS VARCHAR (35) UNIQUE,
PRIMARY KEY (ROLL_NO) );
```

DEFAULT:

The DEFAULT constraint provides a default value to a column when there is no value provided while inserting a record into a table.

```
CREATE TABLE STUDENT( ROLL_NO   INT  NOT NULL,
 STU_NAME VARCHAR (35) NOT NULL,
STU_AGE INT NOT NULL,
EXAM_FEE INT  DEFAULT 10000,
STU_ADDRESS VARCHAR (35) ,
PRIMARY KEY (ROLL_NO) );
```

CHECK:

This constraint is used for specifying range of values for a particular column of a table. When this constraint is being set on a column, it ensures that the specified column must have the value falling in the specified range.

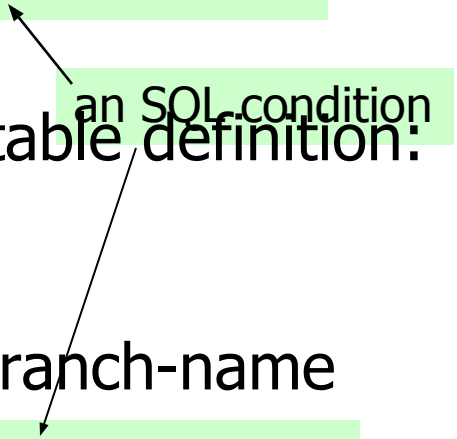
```
CREATE TABLE STUDENT( ROLL_NO   INT  NOT NULL CHECK(ROLL_NO >1000) ,
STU_NAME VARCHAR (35)  NOT NULL,
STU_AGE INT  NOT NULL,
 EXAM_FEE INT DEFAULT 10000,
STU_ADDRESS VARCHAR (35) ,
PRIMARY KEY (ROLL_NO) );
```

Specifying Constraints

- Can have complex conditions in domain check
- …… constraint account-type-test
  check (value in ('Checking', 'Saving'))

- check can be associated with a table definition:

  create table account … …
  check (branch-name in (select branch-name
  from branch))



Adding constraints

```
CREATE TABLE table_name (
column1 datatype [ NULL | NOT NULL ],
column2 datatype [ NULL | NOT NULL ], ...

CONSTRAINT constraint_name CHECK [ NOT FOR REPLICATION ]

(column_name condition) );
```

Eg:

```
CREATE TABLE employees ( employee_id INT NOT NULL, last_name
 VARCHAR(50) NOT NULL, first_name VARCHAR(50), salary MONEY,
CONSTRAINT check_employee_id CHECK (employee_id BETWEEN 1 and 10000) );
```

Assertions

- An assertion is predicate expressing a condition that we wish the database always to satisfy.
- An assertion in SQL-92 takes the form
  create assertion <assertion-name> check <predicate>
- When an assertion is made, the system tests it for validity. This testing may introduce a significant amount of overhead; hence assertions should be used with great care.
- Any predicate allowed in SQL can be used.

Assertion

- Ex for assertion, which enforce a constraint that the *number of boats plus the number of sailors should be less than 100.*
- CREATE ASSERTION smallClub CHECK ((SELECT COUNT (S.sid) FROM Sailors S) + (SELECT COUNT (B.bid) FROM Boats B) < 100);
-

Assertion Example 1

- The sum of all loan amounts for each branch must be less than the sum of all account balances at the branch.

```
create assertion sum-constraint check
    (not exists (select * from branch
            where (select sum(amount) from loan
                where loan.branch-name=branch.branch-name)                    >=
                    (select sum(amount) from account
                where account.branch-name=branch.branch-name) ))
```

Key constraints

PRIMARY KEY:

Primary key uniquely identifies each record in a table. It must have unique values and cannot contain nulls. In the below example the ROLL_NO field is marked as primary key, that means the ROLL_NO field cannot have duplicate and null values.

```
CREATE TABLE STUDENT( ROLL_NO   INT  NOT NULL,
STU_NAME VARCHAR (35)  NOT NULL UNIQUE,
STU_AGE INT NOT NULL,
 STU_ADDRESS VARCHAR (35) UNIQUE,
PRIMARY KEY (ROLL_NO) );
```

Or

```
CREATE TABLE department (
    dpt_no  NUMBER(2) CONSTRAINT pk_department PRIMARY KEY,
    dpt_name VARCHAR2(20)
        CONSTRAINT nn_dpt_name NOT NULL );
or
CREATE TABLE department (
    dpt_no     NUMBER(2),
    dpt_name VARCHAR2(20,
 CONSTRAINT pk_department PRIMARY KEY(dpt_no));
```

FOREIGN KEY:

Foreign keys are the columns of a table that points to the primary key of another table. They act as a cross-reference between tables.

Properties of primary key

- It must contain unique values
- It must not contain null values
- It contains the minimum number of fields to ensure uniqueness
- It must uniquely identify each record in the table

```
CREATE TABLE Enrolled
  (sid CHAR(20)
    cid  CHAR(20),
    grade CHAR(2),
    PRIMARY KEY  (sid))
```

Primary and Candidate Keys in SQL

- Possibly many *candidate keys* (specified using UNIQUE), one of which is chosen as the *primary key*.

```
CREATE TABLE Enrolled
  (sid CHAR(20)
    cid  CHAR(20),
    grade CHAR(2),
    PRIMARY KEY  (sid),
    UNIQUE (cid, grade) )
```

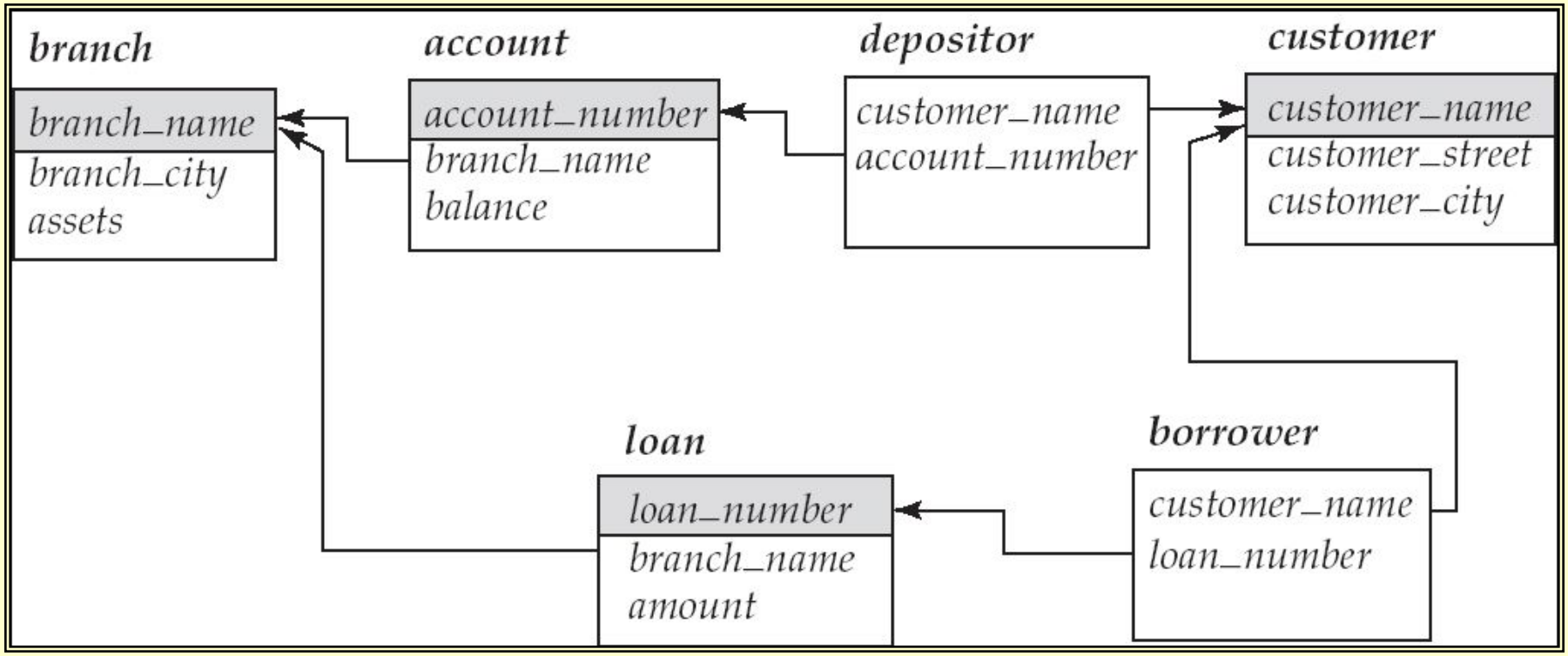
Foreign Keys, Referential Integrity

- *Foreign key* : Set of fields in one relation that is used to `refer' to a tuple in another relation.

- Must correspond to primary key of the second relation. Like a `logical pointer'.

- E.g. *sid* is a foreign key referring to Students:
  - Students (sid:string,name:string.age:number)
  - Enrolled(*sid*: string, *cid*: string, *grade*: string)

Schema Diagram



Cascading Actions in SQL

```
create table account
…..
foreign key (branch_name) references branch
                    on delete cascade
                    on update cascade,
…)
```

- Due to the on delete cascade clauses, if delete of a tuple in branch results in referential-integrity constraint violation, the delete "cascades" to the account relation, deleting the tuple that refers to the branch that was deleted.
- Cascading updates are similar.

The value stored to the branch.branch_name column for any given *account* row must match a value stored in the *branch_name* column in the branch table.

Identifying Foreign Keys

```
CREATE TABLE employee (
emp_dpt_number                NUMBER(2),
CONSTRAINT fk_emp_dpt FOREIGN KEY
   (emp_dpt_number)
     REFERENCES department ON DELETE
   SET NULL
);
```

MAINTAINING REFERRENTIAL INTEGRITY

On Update (Delete) Restrict	Any update/delete made to the *department* table that would delete or change a primary key value will be rejected unless no foreign key references that value in the *employee* table. This is the *default* constraint in Oracle.
On Update (Delete) Cascade	Any update/delete made to the *department* table should be cascaded through to the *employee* table.
On Update (Delete) Set Null	Any values that are updated/deleted in the *department* table cause affected columns in the *employee* table to be set to null.

- CREATE TABLE Enrolled (sid CHAR(20),cid CHAR(20),grade CHAR(10),PRIMARY KEY (sid, cid),FOREIGN KEY (sid) REFERENCES Students **ON DELETE CASCADE ON UPDATE NO ACTION**)
- The options are specified as part of the foreign key declaration. The default option is NO ACTION, which means that the action (DELETE or UPDATE) is to be rejected.

- The CASCADE keyword says that if a Students row is deleted, all Enrolled rows that refer to it are to be deleted as well.
- If the UPDATE clause specified CASCADE, and the *sid* column of a Students row is updated, this update is also carried out in each Enrolled row that refers to the updated Students row.

- If a Students row is deleted, we can switch the enrollment to a `default' student by using ON DELETE SET DEFAULT.
- The default student is specified as part of the definition of the *sid field in Enrolled; for example, sid CHAR(20) DEFAULT `53666'.*
- SQL also allows the use of *null as the default value by specifying ON DELETE SET NULL.*

Add/Remove Constraints

- After you create a table, you can use the Alter Table statement to
  - add or remove a primary key, unique, foreign key, or check constraint

- To drop a table's primary key constraint, just specify the Primary Key keywords:

```
Alter Table Sale
    Drop Primary Key
```

- To drop a unique, foreign key, or check constraint, you must specify the constraint name:

```
Alter Table Sale
    Drop Constraint SaleCustomerFK
```

- To add a new constraint, use the same constraint syntax as in a Create Table statement:

```
Alter Table Sale
    Add Constraint SaleSaleTotChk Check( SaleTot >= 0 )
```