



CS425 – Fall 2013 Boris Glavic Chapter 5: Intermediate SQL

modified from:
Database System Concepts, 6th Ed.
©Silberschatz, Korth and Sudarshan
See www.db-book.com for conditions on re-use



Chapter 5: Intermediate SQL

- Views
- Transactions
- Integrity Constraints
- SQL Data Types and Schemas
- Access Control



Textbook: Chapter 4

CS425 – Fall 2013 – Boris Glavic

5.2

©Silberschatz, Korth and Sudarshan



Views

- In some cases, it is not desirable for all users to see the entire logical model (that is, all the actual relations stored in the database.)
- Consider a person who needs to know an instructors name and department, but not the salary. This person should see a relation described, in SQL, by

```
select ID, name, dept_name
from instructor
```

- A **view** provides a mechanism to hide certain data from the view of certain users.
- Any relation that is not of the conceptual model but is made visible to a user as a “virtual relation” is called a **view**.

CS425 – Fall 2013 – Boris Glavic

5.3

©Silberschatz, Korth and Sudarshan



View Definition

- A view is defined using the **create view** statement which has the form

```
create view v as <query expression >
```

where <query expression> is any legal SQL expression. The view name is represented by v.

- Once a view is defined, the view name can be used to refer to the virtual relation that the view generates.
- View definition is not the same as creating a new relation by evaluating the query expression
 - Rather, a view definition causes the saving of an expression; the expression is substituted into queries using the view.

CS425 – Fall 2013 – Boris Glavic

5.4

©Silberschatz, Korth and Sudarshan



Example Views

- A view of instructors without their salary


```
create view faculty as
select ID, name, dept_name
from instructor
```
- Find all instructors in the Biology department


```
select name
from faculty
where dept_name = 'Biology'
```
- Create a view of department salary totals


```
create view departments_total_salary(dept_name, total_salary) as
select dept_name, sum(salary)
from instructor
group by dept_name;
```

CS425 – Fall 2013 – Boris Glavic

5.5

©Silberschatz, Korth and Sudarshan



Views Defined Using Other Views

- **create view physics_fall_2009 as**


```
select course.course_id, sec_id, building, room_number
from course, section
where course.course_id = section.course_id
and course.dept_name = 'Physics'
and section.semester = 'Fall'
and section.year = '2009';
```
- **create view physics_fall_2009_watson as**

```
select course_id, room_number
from physics_fall_2009
where building = 'Watson';
```

CS425 – Fall 2013 – Boris Glavic

5.6

©Silberschatz, Korth and Sudarshan




View Expansion

- Expand use of a view in a query/another view

```

create view physics_fall_2009_watson as
(select course_id, room_number
from (select course.course_id, building, room_number
      from course, section
      where course.course_id = section.course_id
        and course.dept_name = 'Physics'
        and section.semester = 'Fall'
        and section.year = '2009')
where building = 'Watson');
  
```


CS425 – Fall 2013 – Boris Glavic 5.7 ©Silberschatz, Korth and Sudarshan



Views Defined Using Other Views

- One view may be used in the expression defining another view
- A view relation v_1 is said to *depend directly* on a view relation v_2 if v_2 is used in the expression defining v_1
- A view relation v_1 is said to *depend on* view relation v_2 if either v_1 depends directly to v_2 or there is a path of dependencies from v_1 to v_2
- A view relation v is said to be *recursive* if it depends on itself.


CS425 – Fall 2013 – Boris Glavic 5.8 ©Silberschatz, Korth and Sudarshan



View Expansion

- A way to define the meaning of views defined in terms of other views.
- Let view v_1 be defined by an expression e_1 that may itself contain uses of view relations.
- View expansion of an expression repeats the following replacement step:
 - repeat**
 - Find any view relation v_i in e_1
 - Replace the view relation v_i by the expression defining v_i
 - until** no more view relations are present in e_1
- As long as the view definitions are not recursive, this loop will terminate

CS425 – Fall 2013 – Boris Glavic 5.9 ©Silberschatz, Korth and Sudarshan



Update of a View


- Add a new tuple to *faculty* view which we defined earlier


```

insert into faculty values ('30765', 'Green', 'Music');
  
```

 This insertion must be represented by the insertion of the tuple ('30765', 'Green', 'Music', null) into the *instructor* relation

CS425 – Fall 2013 – Boris Glavic 5.10 ©Silberschatz, Korth and Sudarshan



Some Updates cannot be Translated Uniquely

- ```


create view instructor_info as
select ID, name, building
from instructor, department
where instructor.dept_name= department.dept_name;

```
- ```

insert into instructor_info values ('69987', 'White', 'Taylor');
  
```

 - ▶ which department, if multiple departments in Taylor?
 - ▶ what if no department is in Taylor?
- Most SQL implementations allow updates only on simple views
 - The **from** clause has only one database relation.
 - The **select** clause contains only attribute names of the relation, and does not have any expressions, aggregates, or **distinct** specification.
 - Any attribute not listed in the **select** clause can be set to null
 - The query does not have a **group** by or **having** clause.

CS425 – Fall 2013 – Boris Glavic 5.11 ©Silberschatz, Korth and Sudarshan




And Some Not at All

- ```

create view history_instructors as
select *
from instructor
where dept_name= 'History';

```
- What happens if we insert ('25566', 'Brown', 'Biology', 100000) into *history\_instructors*?


CS425 – Fall 2013 – Boris Glavic 5.12 ©Silberschatz, Korth and Sudarshan



## Materialized Views


- **Materializing a view:** create a physical table containing all the tuples in the result of the query defining the view
- If relations used in the query are updated, the materialized view result becomes out of date
  - Need to **maintain** the view, by updating the view whenever the underlying relations are updated.

CS425 – Fall 2013 – Boris Glavic 5.13 ©Silberschatz, Korth and Sudarshan



## Transactions


modified from:  
Database System Concepts, 6<sup>th</sup> Ed.  
©Silberschatz, Korth and Sudarshan  
See [www.db-book.com](http://www.db-book.com) for conditions on re-use



## Transactions

- Unit of work
- Atomic transaction
  - either fully executed or rolled back as if it never occurred
- Isolation from concurrent transactions
- Transactions begin implicitly
  - Ended by **commit work** or **rollback work**
- But default on most databases: each SQL statement commits automatically
  - Can turn off auto commit for a session (e.g. using API)
  - In SQL:1999, can use: **begin atomic .... end**
    - ▶ Not supported on most databases

CS425 – Fall 2013 – Boris Glavic 5.15 ©Silberschatz, Korth and Sudarshan




## Transactions Example

- Example Atomicity (all-or-nothing)
  - Recall example from the introduction
  - Relation **accounts(accID, cust, type, balance)**
  - A user want to transfer \$100 from his savings (accID = 100) to his checking account (accID= 101)
 

```
UPDATE accounts SET balance = balance - 100 WHERE accID = 100;
UPDATE accounts SET balance = balance + 100 WHERE accID = 101;
```
  - This can cause inconsistencies if the system crashes after the first update (user would loose money)
  - Using a transaction either both or none of the statements are executed

**BEGIN**  
`UPDATE accounts SET balance = balance - 100 WHERE accID = 100;`  
`UPDATE accounts SET balance = balance + 100 WHERE accID = 101;`  
**COMMIT**


CS425 – Fall 2013 – Boris Glavic 5.16 ©Silberschatz, Korth and Sudarshan



## Transactions and Concurrency


- Transactions are also used to isolate concurrent actions of different users
- Recall from the introduction that if several users are modifying the database at the same time that can lead to inconsistencies
- More on that later once we talk about concurrency control

CS425 – Fall 2013 – Boris Glavic 5.17 ©Silberschatz, Korth and Sudarshan



## Integrity Constraints


modified from:  
Database System Concepts, 6<sup>th</sup> Ed.  
©Silberschatz, Korth and Sudarshan  
See [www.db-book.com](http://www.db-book.com) for conditions on re-use



## 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


CS425 – Fall 2013 – Boris Glavic 5.19 ©Silberschatz, Korth and Sudarshan



## Integrity Constraints on a Single Relation

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


CS425 – Fall 2013 – Boris Glavic 5.20 ©Silberschatz, Korth and Sudarshan



## 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** ( $A_1, A_2, \dots, A_m$ )
  - The unique specification states that the attributes  $A_1, A_2, \dots, A_m$  form a candidate key.
  - Candidate keys are permitted to be null (in contrast to primary keys).

CS425 – Fall 2013 – Boris Glavic 5.21 ©Silberschatz, Korth and Sudarshan



## 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'))
);

```


CS425 – Fall 2013 – Boris Glavic 5.22 ©Silberschatz, Korth and Sudarshan



## 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.


CS425 – Fall 2013 – Boris Glavic 5.23 ©Silberschatz, Korth and Sudarshan



## 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**

CS425 – Fall 2013 – Boris Glavic 5.24 ©Silberschatz, Korth and Sudarshan




## 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 without causing constraint violation ?
  - insert father and mother of a person before inserting person
  - OR, set father and mother to null initially, update after inserting all persons (not possible if father and mother attributes declared to be **not null**)
  - OR defer constraint checking (next slide)


CS425 – Fall 2013 – Boris Glavic 5.25 ©Silberschatz, Korth and Sudarshan



## 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


CS425 – Fall 2013 – Boris Glavic 5.26 ©Silberschatz, Korth and Sudarshan



## Indexes and User-Defined Types (UDTs)

modified from:  
Database System Concepts, 6<sup>th</sup> Ed.  
©Silberschatz, Korth and Sudarshan  
See [www.db-book.com](http://www.db-book.com) for conditions on re-use


CS425 – Fall 2013 – Boris Glavic 5.28 ©Silberschatz, Korth and Sudarshan



## Built-in Data Types in SQL

- **date**: Dates, containing a (4 digit) year, month and date
  - Example: **date** '2005-7-27'
- **time**: Time of day, in hours, minutes and seconds.
  - Example: **time** '09:00:30'      **time** '09:00:30.75'
- **timestamp**: date plus time of day
  - Example: **timestamp** '2005-7-27 09:00:30.75'
- **interval**: period of time
  - Example: **interval** '1' day
  - Subtracting a date/time/timestamp value from another gives an interval value
  - Interval values can be added to date/time/timestamp values

CS425 – Fall 2013 – Boris Glavic 5.28 ©Silberschatz, Korth and Sudarshan




## Index Creation

- **create table** *student* (*ID* **varchar** (5), *name* **varchar** (20) **not null**, *dept\_name* **varchar** (20), *tot\_cred* **numeric** (3,0) **default** 0, **primary key** (*ID*))
- **create index** *studentID\_index* on *student*(*ID*)
- Indices are data structures used to speed up access to records with specified values for index attributes
  - e.g. **select** \*  
      **from** *student*  
      **where** *ID* = '12345'

can be executed by using the index to find the required record, without looking at all records of *student*

*More on indices later*

CS425 – Fall 2013 – Boris Glavic 5.29 ©Silberschatz, Korth and Sudarshan




## User-Defined Types

- **create type** construct in SQL creates user-defined type
 

```
create type Dollars as numeric (12,2) final
```

  - **create table** *department* (*dept\_name* **varchar** (20), *building* **varchar** (15), *budget* *Dollars*);

CS425 – Fall 2013 – Boris Glavic 5.30 ©Silberschatz, Korth and Sudarshan




## Domains

- **create domain** construct in SQL-92 creates user-defined domain types
 

```
create domain person_name char(20) not null
```
- Types and domains are similar. Domains can have constraints, such as **not null**, specified on them.
- **create domain** `degree_level varchar(10) constraint degree_level_test check (value in (' Bachelors', ' Masters', ' Doctorate' ));`


CS425 – Fall 2013 – Boris Glavic 5.31 ©Silberschatz, Korth and Sudarshan



## Large-Object Types

- Large objects (photos, videos, CAD files, etc.) are stored as a *large object*:
  - **blob**: binary large object -- object is a large collection of uninterpreted binary data (whose interpretation is left to an application outside of the database system)
  - **clob**: character large object -- object is a large collection of character data
  - When a query returns a large object, a pointer is returned rather than the large object itself.


CS425 – Fall 2013 – Boris Glavic 5.32 ©Silberschatz, Korth and Sudarshan



## Access Control

modified from:  
Database System Concepts, 6<sup>th</sup> Ed.  
©Silberschatz, Korth and Sudarshan  
See [www.db-book.com](http://www.db-book.com) for conditions on re-use

CS425 – Fall 2013 – Boris Glavic 5.34 ©Silberschatz, Korth and Sudarshan



## Access Control


Forms of authorization on parts of the database:

- **Read** - allows reading, but not modification of data.
- **Insert** - allows insertion of new data, but not modification of existing data.
- **Update** - allows modification, but not deletion of data.
- **Delete** - allows deletion of data.

Forms of authorization to modify the database schema

- **Index** - allows creation and deletion of indices.
- **Resources** - allows creation of new relations.
- **Alteration** - allows addition or deletion of attributes in a relation.
- **Drop** - allows deletion of relations.

CS425 – Fall 2013 – Boris Glavic 5.34 ©Silberschatz, Korth and Sudarshan




## Authorization Specification in SQL

- The **grant** statement is used to confer authorization
 

```
grant <privilege list>
on <relation name or view name> to <user list>
```
- <user list> is:
  - a user-id
  - **public**, which allows all valid users the privilege granted
  - A role (more on this later)
- Granting a privilege on a view does not imply granting any privileges on the underlying relations.
- The grantor of the privilege must already hold the privilege on the specified item (or be the database administrator).

CS425 – Fall 2013 – Boris Glavic 5.35 ©Silberschatz, Korth and Sudarshan



## Privileges in SQL

- **select**: allows read access to relation, or the ability to query using the view
  - Example: grant users  $U_1$ ,  $U_2$ , and  $U_3$  **select** authorization on the *instructor* relation:
 

```
grant select on instructor to U1, U2, U3
```
- **insert**: the ability to insert tuples
- **update**: the ability to update using the SQL update statement
- **delete**: the ability to delete tuples.
- **all privileges**: used as a short form for all the allowable privileges

CS425 – Fall 2013 – Boris Glavic 5.36 ©Silberschatz, Korth and Sudarshan

## Revoking Authorization in SQL

- The **revoke** statement is used to revoke authorization.
  - **revoke** <privilege list>
  - **on** <relation name or view name> **from** <user list>
- Example:
  - **revoke select on branch from**  $U_1, U_2, U_3$
- <privilege-list> may be **all** to revoke all privileges the revokee may hold.
- If <revokee-list> includes **public**, all users lose the privilege except those granted it explicitly.
- If the same privilege was granted twice to the same user by different grantees, the user may retain the privilege after the revocation.
- All privileges that depend on the privilege being revoked are also revoked.

CS425 – Fall 2013 – Boris Glavic 5.37 ©Silberschatz, Korth and Sudarshan

## Roles

- **create role** instructor;
- **grant instructor to Amit**;
- Privileges can be granted to roles:
  - **grant select on takes to instructor**;
- Roles can be granted to users, as well as to other roles
  - **create role teaching\_assistant**
  - **grant teaching\_assistant to instructor**;
  - *Instructor inherits all privileges of teaching\_assistant*
- Chain of roles
  - **create role dean**;
  - **grant instructor to dean**;
  - **grant dean to Satoshi**;

CS425 – Fall 2013 – Boris Glavic 5.38 ©Silberschatz, Korth and Sudarshan

## Authorization on Views

- **create view geo\_instructor as**
  - **(select \***
  - **from instructor**
  - **where dept\_name = 'Geology');**
- **grant select on geo\_instructor to geo\_staff**
- Suppose that a *geo\_staff* member issues
  - **select \***
  - **from geo\_instructor**;
- What if
  - *geo\_staff* does not have permissions on *instructor*?
  - creator of view did not have some permissions on *instructor*?

CS425 – Fall 2013 – Boris Glavic 5.39 ©Silberschatz, Korth and Sudarshan

## Other Authorization Features

- **references** privilege to create foreign key
  - **grant reference (dept\_name) on department to Mariano**;
  - why is this required?
- transfer of privileges
  - **grant select on department to Amit with grant option**;
  - **revoke select on department from Amit, Satoshi cascade**;
  - **revoke select on department from Amit, Satoshi restrict**;
- Etc. read text book Section 4.6 for more details we have omitted here.

CS425 – Fall 2013 – Boris Glavic 5.40 ©Silberschatz, Korth and Sudarshan

## Understanding RESTRICT/CASCADE

- Bob grants right X on Y to Alice with grant option
- Alice grants right X on Y to Peter

- **Abandoned right**
  - A right for which there is no justification anymore
- **revoke X on Y from Peter restrict**
  - With restrict fails if it would result in abandoned rights
- **revoke X on Y from Peter cascade**
  - Also revokes rights that would otherwise be abandoned


CS425 – Fall 2013 – Boris Glavic 5.41 ©Silberschatz, Korth and Sudarshan

## Understanding RESTRICT/CASCADE

- Bob grants right X on Y to Alice with grant option
- Alice grants right X on Y to Peter
- Bob grants right X on Y to Peter

- **Abandoned privilege**
  - A privilege for which there is no justification anymore
  - Indirect justifications count
- **revoke X on Y from Peter restrict**
  - Fails: even though there exists additional justification for the privilege.
- **revoke X on Y from Peter cascade**
  - Revokes that right from Peter.
  - Peter still has the right to do X on Y


CS425 – Fall 2013 – Boris Glavic 5.42 ©Silberschatz, Korth and Sudarshan



## Recap


- Views
  - Virtual
  - Materialized
  - Updates
- Integrity Constraints
  - Not null, unique, check
  - Foreign keys: referential integrity
- Access control
  - Users, roles
  - Privileges
  - **GRANT / REVOKE**
- Data types
  - Build-in types, Domains, Large Objects
  - UDTs
  - Indices

CS425 – Fall 2013 – Boris Glavic 5.43 ©Silberschatz, Korth and Sudarshan



## End of Chapter 5


modified from:  
Database System Concepts, 6<sup>th</sup> Ed.  
©Silberschatz, Korth and Sudarshan  
See [www.db-book.com](http://www.db-book.com) for conditions on re-use



## Outline

- Introduction
- Relational Data Model
- Formal Relational Languages (relational algebra)
- **SQL - Advanced**
- Database Design
- Transaction Processing, Recovery, and Concurrency Control
- Storage and File Structures
- Indexing and Hashing
- Query Processing and Optimization


CS425 – Fall 2013 – Boris Glavic 5.45 ©Silberschatz, Korth and Sudarshan



## Figure 4.01

| ID    | name     | dept_name  | tot_cred |
|-------|----------|------------|----------|
| 00128 | Zhang    | Comp. Sci. | 102      |
| 12345 | Shankar  | Comp. Sci. | 32       |
| 19991 | Brandt   | History    | 80       |
| 23121 | Chavez   | Finance    | 110      |
| 44553 | Peltier  | Physics    | 56       |
| 45678 | Levy     | Physics    | 46       |
| 54321 | Williams | Comp. Sci. | 54       |
| 55739 | Sanchez  | Music      | 38       |
| 70557 | Snow     | Physics    | 0        |
| 76543 | Brown    | Comp. Sci. | 58       |
| 76653 | Aoi      | Elec. Eng. | 60       |
| 98765 | Bourikas | Elec. Eng. | 98       |
| 98988 | Tanaka   | Biology    | 120      |


CS425 – Fall 2013 – Boris Glavic 5.46 ©Silberschatz, Korth and Sudarshan



## Figure 4.02

| ID    | course_id | sec_id | semester | year | grade |
|-------|-----------|--------|----------|------|-------|
| 00128 | CS-101    | 1      | Fall     | 2009 | A     |
| 00128 | CS-347    | 1      | Fall     | 2009 | A-    |
| 12345 | CS-101    | 1      | Fall     | 2009 | C     |
| 12345 | CS-190    | 2      | Spring   | 2009 | A     |
| 12345 | CS-315    | 1      | Spring   | 2010 | A     |
| 12345 | CS-347    | 1      | Fall     | 2009 | A     |
| 19991 | HIS-351   | 1      | Spring   | 2010 | B     |
| 23121 | FIN-201   | 1      | Spring   | 2010 | C+    |
| 44553 | PHY-101   | 1      | Fall     | 2009 | B-    |
| 45678 | CS-101    | 1      | Fall     | 2009 | F     |
| 45678 | CS-101    | 1      | Spring   | 2010 | B+    |
| 45678 | CS-319    | 1      | Spring   | 2010 | B     |
| 54321 | CS-101    | 1      | Fall     | 2009 | A-    |
| 54321 | CS-190    | 2      | Spring   | 2009 | B+    |
| 55739 | MU-199    | 1      | Spring   | 2010 | A-    |
| 76543 | CS-101    | 1      | Fall     | 2009 | A     |
| 76543 | CS-319    | 2      | Spring   | 2010 | A     |
| 76653 | EE-181    | 1      | Spring   | 2009 | C     |
| 98765 | CS-101    | 1      | Fall     | 2009 | C-    |
| 98765 | CS-315    | 1      | Spring   | 2010 | B     |
| 98988 | BIO-101   | 1      | Summer   | 2009 | A     |
| 98988 | BIO-301   | 1      | Summer   | 2010 | null  |

CS425 – Fall 2013 – Boris Glavic 5.47 ©Silberschatz, Korth and Sudarshan



## Figure 4.03

| ID    | name     | dept_name  | tot_cred | course_id | sec_id | semester | year | grade |
|-------|----------|------------|----------|-----------|--------|----------|------|-------|
| 00128 | Zhang    | Comp. Sci. | 102      | CS-101    | 1      | Fall     | 2009 | A     |
| 00128 | Zhang    | Comp. Sci. | 102      | CS-347    | 1      | Fall     | 2009 | A-    |
| 12345 | Shankar  | Comp. Sci. | 32       | CS-101    | 1      | Fall     | 2009 | C     |
| 12345 | Shankar  | Comp. Sci. | 32       | CS-190    | 2      | Spring   | 2009 | A     |
| 12345 | Shankar  | History    | 32       | CS-315    | 1      | Spring   | 2010 | A     |
| 12345 | Shankar  | Finance    | 32       | CS-347    | 1      | Fall     | 2009 | A     |
| 19991 | Brandt   | Music      | 80       | HIS-351   | 1      | Spring   | 2010 | B     |
| 23121 | Chavez   | Physics    | 110      | FIN-201   | 1      | Spring   | 2010 | C+    |
| 44553 | Peltier  | Physics    | 56       | PHY-101   | 1      | Fall     | 2009 | B-    |
| 45678 | Levy     | Physics    | 46       | CS-101    | 1      | Fall     | 2009 | F     |
| 45678 | Levy     | Physics    | 46       | CS-101    | 1      | Spring   | 2010 | B+    |
| 45678 | Levy     | Physics    | 46       | CS-319    | 1      | Spring   | 2010 | B     |
| 54321 | Williams | Comp. Sci. | 54       | CS-101    | 1      | Fall     | 2009 | A-    |
| 54321 | Williams | Comp. Sci. | 54       | CS-190    | 2      | Spring   | 2009 | B+    |
| 55739 | Sanchez  | Music      | 38       | MU-199    | 1      | Spring   | 2010 | A-    |
| 76543 | Brown    | Comp. Sci. | 58       | CS-101    | 1      | Fall     | 2009 | A     |
| 76543 | Brown    | Comp. Sci. | 58       | CS-319    | 2      | Spring   | 2010 | A     |
| 76653 | Aoi      | Elec. Eng. | 60       | EE-181    | 1      | Spring   | 2009 | C     |
| 98765 | Bourikas | Elec. Eng. | 98       | CS-101    | 1      | Fall     | 2009 | C-    |
| 98765 | Bourikas | Elec. Eng. | 98       | CS-315    | 1      | Spring   | 2010 | B     |
| 98988 | Tanaka   | Biology    | 120      | BIO-101   | 1      | Summer   | 2009 | A     |
| 98988 | Tanaka   | Biology    | 120      | BIO-301   | 1      | Summer   | 2010 | null  |

CS425 – Fall 2013 – Boris Glavic 5.48 ©Silberschatz, Korth and Sudarshan



