The Relational Model

CMU SCS 15-415/615
C. Faloutsos – A. Pavlo
Lecture #3
R & G, Chap. 3

Why Study the Relational Model?

- **Most widely used model.**
  - Vendors: IBM/Informix, Microsoft, Oracle, Sybase, etc.
- **“Legacy systems” in older models**
  - e.g., IBM’s IMS
- **Object-oriented concepts have merged in**
  - *object-relational model*
    - Informix->IBM DB2, Oracle

Roadmap

- **Introduction**
- **Integrity constraints (IC)**
- **Enforcing IC**
- **Querying Relational Data**
- **ER to tables**
- **Intro to Views**
- **Destroying/altering tables**

Relational Database: Definitions

- **Relational database**: a set of *relations*
- (relation = table)
- specifically

```
<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>login</th>
<th>age</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>5566</td>
<td>Jones</td>
<td>jones</td>
<td>18</td>
<td>3.5</td>
</tr>
<tr>
<td>5568</td>
<td>Smith</td>
<td>smith</td>
<td>15</td>
<td>3.2</td>
</tr>
<tr>
<td>5600</td>
<td>Smith</td>
<td>smith</td>
<td>15</td>
<td>3.8</td>
</tr>
</tbody>
</table>
```

Faloutsos - Pavlo, 15-415/615
Relational Database: Definitions

- **Relation**: made up of 2 parts:
  - **Schema**: specifies name of relation, plus name and type of each column.
  - **Instance**: a table, with rows and columns.
    - \#rows = **cardinality**
    - \#fields = **degree / arity**

Ex: Instance of Students Relation

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>login</th>
<th>age</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>53666</td>
<td>Jones</td>
<td>jones@cs</td>
<td>18</td>
<td>3.4</td>
</tr>
<tr>
<td>53688</td>
<td>Smith</td>
<td>smith@cs</td>
<td>18</td>
<td>3.2</td>
</tr>
<tr>
<td>53650</td>
<td>Smith</td>
<td>smith@math</td>
<td>19</td>
<td>3.8</td>
</tr>
</tbody>
</table>

- Cardinality = 3, arity = 5,
- all rows distinct
- Q: do values in a column need to be distinct?

SQL - A language for Relational DBs

- **SQL** (a.k.a. “Sequel”), standard language
- **Data Definition Language (DDL)**
  - create, modify, delete relations
  - specify constraints
  - administer users, security, etc.
  - E.g.: `create table student (ssn fixed, name char(20));`

*Structured Query Language*
Data Manipulation Language (DML)
- Specify queries to find tuples that satisfy criteria
- add, modify, remove tuples

select * from student ;
update takes set grade=4
  where name='smith'
    and cid = 'db';

SQL Overview
- CREATE TABLE <name> ( <field> <domain>, ... )
- INSERT INTO <name> ( <field names> )
  VALUES ( <field values> )
- DELETE FROM <name>
  WHERE <condition>

SQL Overview
- UPDATE <name>
  SET <field name> = <value>
  WHERE <condition>
- SELECT <fields>
  FROM <name>
  WHERE <condition>

Creating Relations in SQL
- Creates the Students relation.

CREATE TABLE Students
  ( sid CHAR(20),
    name CHAR(20),
    login CHAR(10),
    age INTEGER,
    gpa FLOAT )
Creating Relations in SQL

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

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

Table Creation (continued)

- **Another example:**

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

Adding and Deleting Tuples

- **Can insert a single tuple using:**

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

Adding and Deleting Tuples

- 'mass'-delete (**all** Smiths!):

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

- Introduction
- Integrity constraints (IC)
- Enforcing IC
- Querying Relational Data
- ER to tables
- Intro to Views
- Destroying/altering tables

(Motivation: )

- In flat files, how would you check for duplicate ssn, in a student file?
- (horror stories, if ssn is duplicate?)

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>login</th>
<th>age</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>53666</td>
<td>Jones</td>
<td>jones@cs</td>
<td>18</td>
<td>3.4</td>
</tr>
<tr>
<td>53668</td>
<td>Smith</td>
<td>smith@cs</td>
<td>18</td>
<td>3.2</td>
</tr>
<tr>
<td>53650</td>
<td>Smith</td>
<td>smith@math</td>
<td>19</td>
<td>3.8</td>
</tr>
</tbody>
</table>

Keys

- Keys help associate tuples in different relations
- Keys are one form of integrity constraint (IC)

### Enrolled

<table>
<thead>
<tr>
<th>sid</th>
<th>cid</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>53666</td>
<td>15-101</td>
<td>C</td>
</tr>
<tr>
<td>53666</td>
<td>18-203</td>
<td>B</td>
</tr>
<tr>
<td>53650</td>
<td>15-112</td>
<td>A</td>
</tr>
<tr>
<td>53666</td>
<td>15-105</td>
<td>B</td>
</tr>
</tbody>
</table>

### Students

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>login</th>
<th>age</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>53666</td>
<td>Jones</td>
<td>jones@cs</td>
<td>18</td>
<td>3.4</td>
</tr>
<tr>
<td>53688</td>
<td>Smith</td>
<td>smith@cs</td>
<td>18</td>
<td>3.2</td>
</tr>
<tr>
<td>53650</td>
<td>Smith</td>
<td>smith@math</td>
<td>19</td>
<td>3.8</td>
</tr>
</tbody>
</table>

*FOREIGN Key*

*PRIMARY Key*
Primary Keys

• A set of fields is a **superkey** if:
  – No two distinct tuples can have same values in all key fields

• A set of fields is a **key** for a relation if:
  – Minimal superkey

Student (ssn, name, address)
- \{ssn, name\}: superkey
- \{ssn\}: superkey, AND key
- \{name\}: not superkey

Primary Keys

• **what if >1 key for a relation?**
  – One of the keys is chosen (by DBA) to be the **primary key**. Other keys are called **candidate** keys.
  – Q: example of >1 superkeys?
    - A1: student: \{ssn\}, \{student-id\}, \{driving license\}, \{state\}
    - A2: Employee: \{ssn\}, \{phone\}, \{room\}
    - A3: computer: \{mac-address\}, \{serial\}
Primary Keys

- E.g.
  - `sid` is a key for Students.
  - What about `name`?
  - The set `{sid, gpa}` is a superkey.

Syntax:

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

Drill:

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

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

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

Q: what does this mean?

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

Primary and Candidate Keys in SQL

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

vs.

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

“Students can take only one course, and no two students in a course receive the same grade.”

Foreign Keys

Students

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>login</th>
<th>age</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>53666</td>
<td>Jones</td>
<td>jones@cs</td>
<td>18</td>
<td>3.4</td>
</tr>
<tr>
<td>53688</td>
<td>Smith</td>
<td>smith@cs</td>
<td>18</td>
<td>3.2</td>
</tr>
<tr>
<td>53650</td>
<td>Smith</td>
<td>smith@math</td>
<td>19</td>
<td>3.8</td>
</tr>
</tbody>
</table>

Enrolled

<table>
<thead>
<tr>
<th>sid</th>
<th>cid</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>53666</td>
<td>15-101</td>
<td>C</td>
</tr>
<tr>
<td>53666</td>
<td>18-203</td>
<td>B</td>
</tr>
<tr>
<td>53650</td>
<td>15-112</td>
<td>A</td>
</tr>
<tr>
<td>53666</td>
<td>15-105</td>
<td>B</td>
</tr>
</tbody>
</table>

Foreign Keys, Referential Integrity

- **Foreign key**: Set of fields `refering` to a tuple in another relation.
  - Must correspond to the primary key of the other relation.
  - Like a `logical pointer`.
- **foreign key constraints enforce referential integrity** (i.e., no dangling references.)
### Foreign Keys in SQL

Example: Only existing students may enroll for courses.
- *sid* is a foreign key referring to *Students*:

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

### Roadmap

- Introduction
- Integrity constraints (IC)
- Enforcing IC
- Querying Relational Data
- ER to tables
- Intro to Views
- Destroying/altering tables

### Enforcing Referential Integrity

**Subtle issues:**
- What should be done if an Enrolled tuple with a non-existent student id is inserted?
Enforcing Referential Integrity

• Subtle issues:
  • What should be done if an Enrolled tuple with a non-existent student id is inserted? *(Reject it!)*

<table>
<thead>
<tr>
<th>sid</th>
<th>cid</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>53666</td>
<td>15-101</td>
<td>C</td>
</tr>
<tr>
<td>53666</td>
<td>18-203</td>
<td>B</td>
</tr>
<tr>
<td>53650</td>
<td>15-112</td>
<td>A</td>
</tr>
<tr>
<td>53666</td>
<td>15-105</td>
<td>B</td>
</tr>
</tbody>
</table>

Students

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>login</th>
<th>age</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>53666</td>
<td>Jones</td>
<td>jones@cs</td>
<td>18</td>
<td>3.4</td>
</tr>
<tr>
<td>53688</td>
<td>Smith</td>
<td>smith@cs</td>
<td>18</td>
<td>3.2</td>
</tr>
<tr>
<td>53650</td>
<td>Smith</td>
<td>smith@math</td>
<td>19</td>
<td>3.8</td>
</tr>
</tbody>
</table>

Enforcing Referential Integrity

• Subtle issues, cont’d:
  • What should be done if a Student’s tuple is deleted?
    – Also delete all Enrolled tuples that refer to it?
    –Disallow deletion of a Students tuple that is referred to?
    – Set sid in Enrolled tuples that refer to it to a default sid?
    – (In SQL, also: Set sid in Enrolled tuples that refer to it to a special value *null*, denoting ‘unknown’ or ‘inapplicable’.)

Enforcing Referential Integrity

• Similar issues arise if primary key of Students tuple is updated.
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.

Where do ICs Come From?

- the application!
Where do ICs Come From?

- Subtle point: We can check a database instance to see if an IC is violated, but we can NEVER infer that an IC is true by looking at an instance.
  - An IC is a statement about all possible instances!
  - Eg., name is not a key,
  - but the assertion that sid is a key is given to us.

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>login</th>
<th>age</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>33666</td>
<td>Jones</td>
<td>jones@ics</td>
<td>18</td>
<td>3.4</td>
</tr>
<tr>
<td>33688</td>
<td>Smith</td>
<td>smith@ics</td>
<td>18</td>
<td>3.2</td>
</tr>
<tr>
<td>33650</td>
<td>Smith</td>
<td>smith@math</td>
<td>19</td>
<td>3.8</td>
</tr>
</tbody>
</table>

Where do ICs Come From?
- Key and foreign key ICs are the most common; more general ICs supported too.

Roadmap

- Introduction
- Integrity constraints (IC)
- Enforcing IC
- Querying Relational Data
- ER to tables
- Intro to Views
- Destroying/altering tables

ER to tables outline:

- strong entities
- weak entities
- (binary) relationships
  - 1-to-1, 1-to-many, etc
  - total/partial participation
- ternary relationships
- ISA-hierarchies
- aggregation
Logical DB Design: ER to Relational

• (strong) entity sets to tables.

CREATE TABLE Employees
    (ssn CHAR(11),
     name CHAR(20),
     lot INTEGER,
     PRIMARY KEY (ssn))

Relationship Sets to Tables

Many-to-many:

CREATE TABLE Works_In
    (ssn CHAR(11),
     did CHAR(11),
     since DATE,
     PRIMARY KEY (ssn, did))
Relationship Sets to Tables

- key of many-to-many relationships:
  - Keys from participating entity sets (as foreign keys).

```
CREATE TABLE Works_In(
    ssn CHAR(11),
    did INTEGER,
    since DATE,
    PRIMARY KEY (ssn, did),
    FOREIGN KEY (ssn)
    REFERENCES Employees,
    FOREIGN KEY (did)
    REFERENCES Departments
)
```

<table>
<thead>
<tr>
<th>Ssn</th>
<th>did</th>
<th>since</th>
</tr>
</thead>
<tbody>
<tr>
<td>123-22-6666</td>
<td>51</td>
<td>1/1/91</td>
</tr>
<tr>
<td>123-22-6666</td>
<td>56</td>
<td>3/3/93</td>
</tr>
<tr>
<td>233-31-5363</td>
<td>51</td>
<td>2/2/92</td>
</tr>
</tbody>
</table>

Review: Key Constraints in ER

- 1-to-many:

```
CREATE TABLE Departments(
    did INTEGER,
    dname VARCHAR(20),
    budget REAL,
    PRIMARY KEY (did)
)
```

```
CREATE TABLE Employees(
    ssn CHAR(11),
    name VARCHAR(20),
    lot DATE,
    PRIMARY KEY (ssn)
)
```

```
CREATE TABLE Manages(
    ssn CHAR(11),
    did INTEGER,
    since DATE,
    PRIMARY KEY (ssn, did)
)
```

ER to tables - summary of basics

- strong entities:
  - key -> primary key
- (binary) relationships:
  - get keys from all participating entities - pr. key:
    - 1-to-1 -> either key (other: ‘cand. key’)
    - 1-to-N -> the key of the ‘N’ part
    - M-to-N -> both keys
A subtle point (1-to-many)

Translating ER with Key Constraints

CREATE TABLE Manages(
    ssn CHAR(11),
    did INTEGER,
    since DATE,
    PRIMARY KEY (did),
    FOREIGN KEY (ssn) REFERENCES Employees,
    FOREIGN KEY (did) REFERENCES Departments)

CREATE TABLE Departments(
    did INTEGER,
    dname CHAR(20),
    budget REAL,
    PRIMARY KEY (did),
    FOREIGN KEY (did) REFERENCES Departments)

Two-table-solution

CREATE TABLE Dept_Mgr(
    ssn CHAR(11),
    did INTEGER,
    since DATE,
    dname CHAR(20),
    budget REAL,
    PRIMARY KEY (did),
    FOREIGN KEY (ssn) REFERENCES Employees)

Single-table-solution

Vs.
Pros and cons?

Drill:

What if the toy department has no manager (yet)?

A: one-table solution can not handle that. (ie., helps enforce ‘thick arrow’ – see next)

CREATE TABLE Dept_Mgr(
did INTEGER,
dname CHAR(20),
budget REAL,
ssn CHAR(11),
since DATE,
PRIMARY KEY (did),
FOREIGN KEY (ssn)
REFERENCES Employees)

ER to tables outline:

✓ strong entities
• weak entities
• (binary) relationships
  ✓ 1-to-1, 1-to-many, etc
    – total/partial participation
• ternary relationships
• ISA-hierarchies
• aggregation
Review: Participation Constraints

- Does every department have a manager?
  - If so, this is a participation constraint: the participation of Departments in Manages is said to be total (vs. partial).
- Every did value in Departments table must appear in a row of the Manages table (with a non-null ssn value!)

Participation Constraints in SQL

- We can capture participation constraints involving one entity set in a binary relationship, but little else (without resorting to CHECK constraints).

```sql
CREATE TABLE Dept_Mgr(
    did INTEGER,
    dname CHAR(20),
    budget REAL,
    ssn CHAR(11) NOT NULL,
    since DATE,
    PRIMARY KEY (did),
    FOREIGN KEY (ssn) REFERENCES Employees,
    ON DELETE NO ACTION
)
```

- Total participation ('no action' -> do NOT do the delete)
- Ie, a department MUST have a manager

```sql
CREATE TABLE Dept_Mgr(
    did INTEGER,
    dname CHAR(20),
    budget REAL,
    ssn CHAR(11) NOT NULL,
    since DATE,
    PRIMARY KEY (did),
    FOREIGN KEY (ssn) REFERENCES Employees,
    ON DELETE NO ACTION
)
```

- Partial participation, ie, a department may be headless

```sql
CREATE TABLE Dept_Mgr(
    did INTEGER,
    dname CHAR(20),
    budget REAL,
    ssn CHAR(11) NOT NULL,
    since DATE,
    PRIMARY KEY (did),
    FOREIGN KEY (ssn) REFERENCES Employees,
    ON DELETE SET NULL
)
```
ER to tables outline:

- strong entities
- weak entities
- (binary) relationships
  - 1-to-1, 1-to-many, etc
  - total/partial participation
- ternary relationships
- ISA-hierarchies
- aggregation

Review: Weak Entities

A weak entity can be identified uniquely only by considering the primary key of another (owner) entity.

- Owner entity set and weak entity set must participate in a one-to-many relationship set (1 owner, many weak entities).
- Weak entity set must have total participation in this identifying relationship set.

How to turn ‘Dependents’ into a table?

Translating Weak Entity Sets

Weak entity set and identifying relationship set are translated into a single table.

```
CREATE TABLE Dep_Policy (
    dname CHAR(20),
    age INTEGER,
    cost REAL,
    ssn CHAR(11) NOT NULL,
    PRIMARY KEY (dname, ssn),
    FOREIGN KEY (ssn) REFERENCES Employees,
    ON DELETE CASCADE
)
```
Translating Weak Entity Sets

- Weak entity set and identifying relationship set are translated into a single table.
  - When the owner entity is deleted, all owned weak entities must also be deleted (-> 'CASCADE')

CREATE TABLE Dep_Policy (  
  dname CHAR(20),  
  age INTEGER,  
  cost REAL,  
  ssn CHAR(11) NOT NULL,  
  PRIMARY KEY (dname, ssn),  
  FOREIGN KEY (ssn) REFERENCES Employees,  
  ON DELETE CASCADE)

Review: ISA Hierarchies

- **Overlap constraints**: Can Joe be an Hourly_Emps as well as a Contract_Emps entity? (Allowed/disallowed)
- **Covering constraints**: Does every Employees entity also have to be an Hourly_Emps or a Contract_Emps entity? (Yes/no)

ER to tables outline:

- **strong entities**
- **weak entities**
- (binary) relationships
  - 1-to-1, 1-to-many, etc
  - total/partial participation
- ternary relationships
- ISA-hierarchies
- aggregation

Drill:

- **What would you do?**

```sql
CREATE TABLE Dep_Policy (  
  dname CHAR(20),  
  age INTEGER,  
  cost REAL,  
  ssn CHAR(11) NOT NULL,  
  PRIMARY KEY (dname, ssn),  
  FOREIGN KEY (ssn) REFERENCES Employees,  
  ON DELETE CASCADE)
```
Translating ISA Hierarchies to Relations

- General approach: 3 relations: Employees, Hourly_Emps and Contract_Emps.
  - how many times do we record an employee?
  - what to do on deletion?
  - how to retrieve all info about an employee?

\[
\begin{align*}
\text{EMP (ssn, name, lot)} \\
\text{H_EMP(ssn, h_wg, h_wk)} \\
\text{CONTR(ssn, cid)}
\end{align*}
\]

- Alternative: Just Hourly_Emps and Contract_Emps.
  - Hourly_Emps: \( ssn, name, lot, hourly_wages, hours_worked \).
  - Each employee must be in one of these two subclasses.

\[
\begin{align*}
\text{EMP (ssn, name, lot)} \\
\text{H_EMP(ssn, h_wg, h_wk, name, lot)} \\
\text{CONTR(ssn, cid, name, lot)}
\end{align*}
\]

Notice: ‘black’ is gone!

Not in book – why NOT 1 table + nulls?

- all hourly contractors

Not in book – why NOT 1 table + nulls?

- all hourly contractors
ER to tables outline:

- strong entities
- weak entities
- (binary) relationships
  - 1-to-1, 1-to-many, etc
  - total/partial participation
- ternary relationships
- ISA-hierarchies
- aggregation

Ternary relationships; aggregation

- rare
- keep keys of all participating entity sets

(or: avoid such situations:
  break into 2-way relationships or
  add an auto-generated key)

Roadmap

- Introduction
- Integrity constraints (IC)
- Enforcing IC
- Querying Relational Data
- ER to tables
- Intro to Views
- Destroying/altering tables

Views

- Virtual tables
  CREATE VIEW YoungActiveStudents
  (name,grade)
  AS SELECT S.name, E.grade
  FROM Students S, Enrolled E
  WHERE S.sid=E.sid and S.age<21
- DROP VIEW
Views and Security

- **DBA**: grants authorization to a view for a user
- user can only see the view - nothing else

Table changes

- DROP TABLE
- ALTER TABLE, e.g.
  
  ```
  ALTER TABLE students
    ADD COLUMN maiden-name CHAR(10)
  ```

Roadmap

- Introduction
- Integrity constraints (IC)
- Enforcing IC
- Querying Relational Data
- ER to tables
- Intro to Views
- Destroying/altering tables

Relational Model: Summary

- A **tabular** representation of data.
- Simple and intuitive; widely used
- Integrity constraints can be specified by the DBA, based on customer specs. DBMS checks for violations.
  - Two important ICs: **primary** and **foreign** keys
  - also: not null, unique
  - In addition, we **always** have domain constraints.
- **Mapping** from ER to Relational is (fairly) straightforward:
ER to tables - summary of basics

- **strong entities:**
  - key -> primary key

- **(binary) relationships:**
  - get keys from all participating entities - pr. key:
    - 1:1 -> either key
    - 1:N -> the key of the 'N' part
    - M:N -> both keys

- **weak entities:**
  - strong key + partial key -> primary key
  - ...... ON DELETE CASCADE

ER to tables - summary of advanced

- **total/partial participation:**
  - NOT NULL; ON DELETE NO ACTION

- **ternary relationships:**
  - get keys from all; decide which one(s) -> prim. key

- **aggregation: like relationships**

- **ISA:**
  - 2 tables ('total coverage')
  - 3 tables (most general)