計算機システム概論
データベースとSQL
2011/5/13

門林雄基

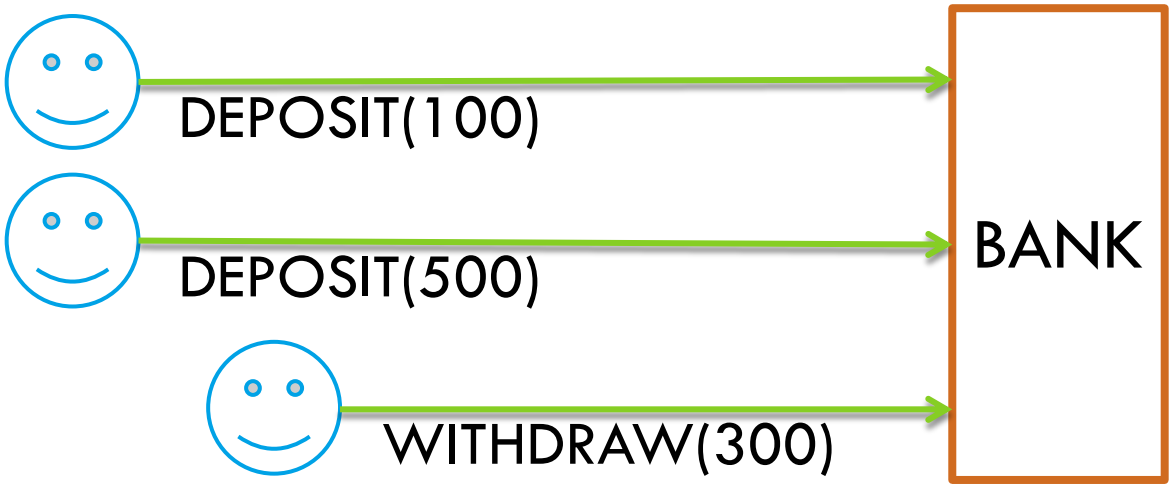
NAIST 奈良先端科学技術大学院大学

講義のポイント

- データへのアクセスが集中する状況で
  データの一貫性を保つには？
- 突然電源が落ちてもデータの一貫性を
  保つには？
- データベースの基本的な考え方とは？
- データベースの操作はどうやるのか？

データの一貫性の維持:
並行制御とトランザクション

データの一貫性の維持



- → 並行制御が必要不可欠

複数プログラム動作時の競合問題

- deposit(N, x)
```
{
  a = read(N);
  a = a + x;
  write(N, a);
}
```

- withdraw(N, x)
```
{
  a = read(N);
  a = a – x;
  write(N, a);
}
```

銀行口座への預け入れと引き出しが、もし同時に実行されたら？
このような競合する可能性のある操作を正しく行うには？

セマフォ (Semaphore)

- P() または wait()
  - セマフォが正の整数になるまで待ち、1減算
- V() または post()
  - セマフォを1増やし、
    待っているプロセス（またはスレッド）を起こす

競合問題のセマフォによる解決

- initialize(S, 1);

- deposit(N, x)
```
{
  wait(S);
  a = read(N);
  a = a + x;
  write(N, a);
  post(S);
}
```

- withdraw(N, x)
```
{
  wait(S);
  a = read(N);
  a = a – x;
  write(N, a);
  post(S);
}
```

Critical section

セマフォにより排他制御 (Mutual exclusion) を実現

並行制御の記述

さまざまな記述法があるが、互いを記述可能

- Semaphore
- Mutex
- Lock
- Monitor
- Message queue
- Condition variable

問題:

- ディスクに書き込んでいる途中に停電したら？

トランザクション

- 一連の処理をグループ化し、以下の特性を実現する:
  - Atomicity:
    一連の処理が完了しているか、まったく処理していないか
  - Consistency:
    トランザクションの前後で、一貫性のある状態に
  - Serializability (Isolation):
    並行トランザクションを直列化可能
  - Durable:
    結果が消えることがない

- “ACID properties”

トランザクションの基本操作

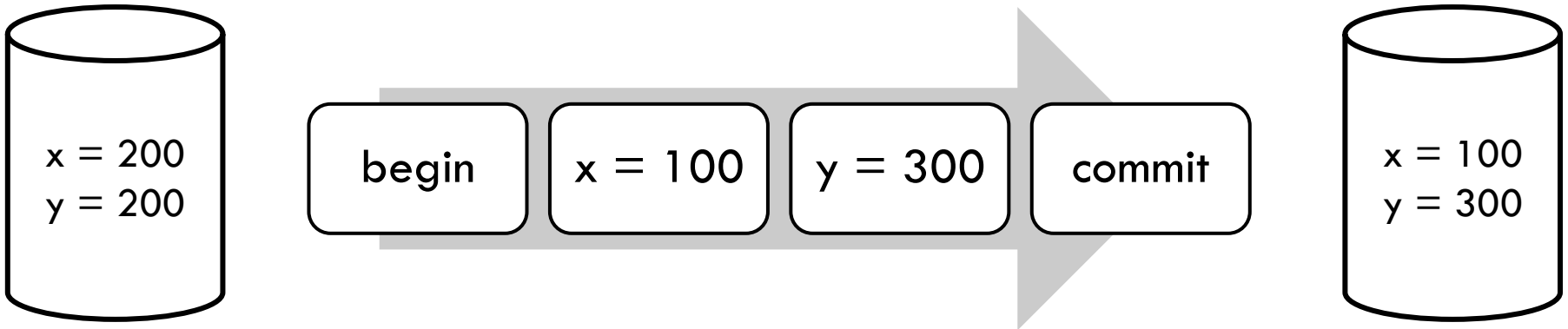
- Begin: トランザクションの開始
- Commit: トランザクションが終了したら、commit する
- Rollback: 途中で故障/不都合が起きたら、トランザクション開始時点の状態に rollback する

- 例: 銀行口座 x から y への送金
- Begin transaction
  - x = x - 100;
  - y = y + 100;
- Commit

トランザクションの実現

- 基本的な考え方: ディスクへの複数回の書き込みを（意味的に）一回にする

- ディスク上に、"write-ahead" log を持つ。
- ログにはデータの更新をすべて記録する。
- 単一トランザクション内のデータの更新がすべてログに記録できれば、commit を書き込む。そのあと変更内容をディスクに書き込む。



Write-ahead logging

- 1. 新しい x の残高をログに書き込む
- 2. 新しい y の残高をログに書き込む
- 3. commit を書き込む
- 4. x をディスクに書き込む
- 5. y をディスクに書き込む
- 6. ログの領域を解放する
- 1 のあとクラッシュした場合: 送金はなされていない
- 2 のあとクラッシュした場合: 同じ
- 3, 4, 5 のあとクラッシュした場合: commit が書いてあるところまで、ログの内容をディスクに反映する

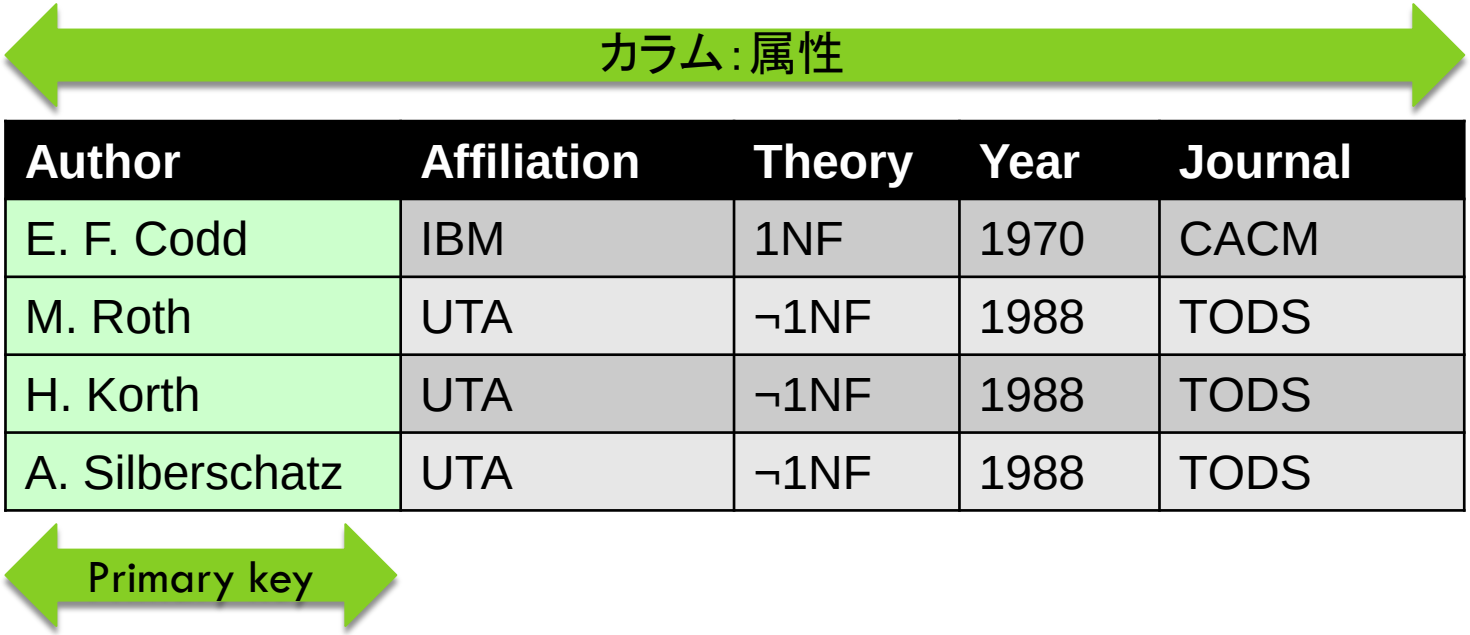
Write-ahead logging

- commit を書き込んでいる最中にクラッシュしたら？
  - 送金はなされていない
  - 工夫: commit はディスクの1セクタとし、CRC（巡回冗長符号）を末尾に付ける
- Commitの書き込み = atomic

データベースの計算モデル：関係代数

関係代数

- 関係代数 (Relational Algebra)
- データベースにおける計算モデル
- 「データベースとは関係の集合である」



Author	Affiliation	Theory	Year	Journal
E. F. Codd	IBM	1NF	1970	CACM
M. Roth	UTA	¬1NF	1988	TODS
H. Korth	UTA	¬1NF	1988	TODS
A. Silberschatz	UTA	¬1NF	1988	TODS

関係データベースの例

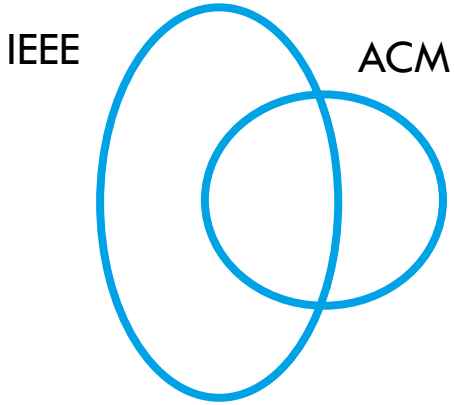
Journal	JournalName	Society
CACM	Communication of ACM	ACM
TODS	Transactions on Database Systems	ACM

Author	Affiliation	Theory	Year	Journal
E. F. Codd	IBM	1NF	1970	CACM
M. Roth	UTA	¬1NF	1988	TODS
H. Korth	UTA	¬1NF	1988	TODS
A. Silberschatz	UTA	¬1NF	1988	TODS

Affiliation	AffiliationName	Country
IBM	IBM Research Laboratory	Etats-Unis
UTA	University of Texas at Austin	Etats-Unis

関係演算子 ∪ (set union)

- ACM_PAPER ∪ IEEE_PAPER



ACM_PAPER

Author	Affiliation	Theory	Year	Journal
E. F. Codd	IBM	1NF	1970	CACM
M. Roth	UTA	¬1NF	1988	TODS
H. Korth	UTA	¬1NF	1988	TODS
A. Silberschatz	UTA	¬1NF	1988	TODS

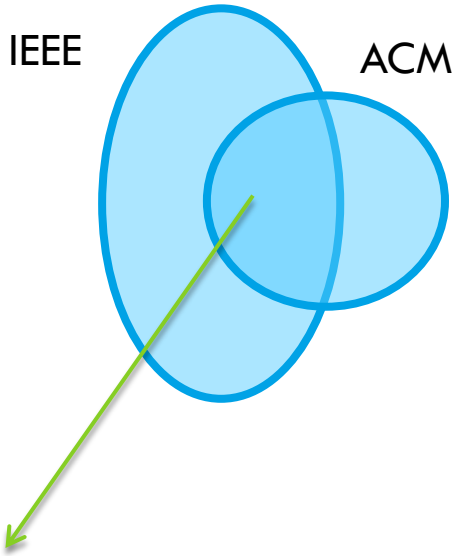
IEEE_PAPER

Author	Affiliation	Theory	Year	Journal
S. Ceri	PDM	SQL	1985	TOSE
G. Gottlob	PDM	SQL	1985	TOSE

同一属性であるので
Union compatible
である

関係演算子 ∩ (set intersection)

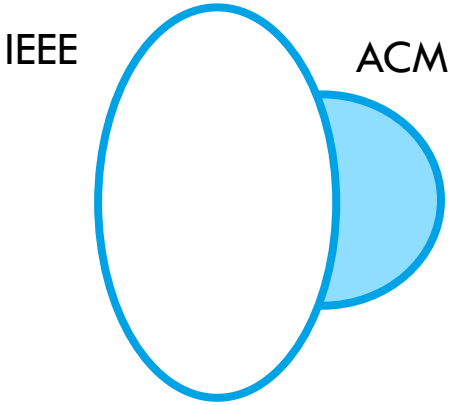
- ACM_PAPER ∩ IEEE_PAPER



IEEE/ACM Trans. on Networking

関係演算子 – (set difference)

- ACM_PAPER – IEEE_PAPER



- では次に
  ACMにしか投稿していない著者を探したい

関係演算子 π (projection)

- $\pi_{author}(IEEE\_PAPER)$
- すべてのIEEE Authorを返す – ただし重複は除く

IEEE_PAPER

Author	Affiliation	Theory	Year	Journal
S. Ceri	PDM	SQL	1985	TOSE
G. Gottlob	PDM	SQL	1985	TOSE

- ACMにしか投稿していない著者を求めるには:
  $\pi_{author}(ACM\_PAPER) - \pi_{author}(IEEE\_PAPER)$

- では次に、IBM からの投稿論文だけ探したい

関係演算子 σ (selection)

- Σ (sigma) の小文字 σ

- IBMからのIEEE投稿論文
- $\sigma_{affiliation='IBM'}(IEEE\_PAPER)$

- 1990年以前のIBMからの論文投稿者
- $\pi_{author}(\sigma_{affiliation='IBM' \wedge year < 1990}(IEEE\_PAPER \cup ACM\_PAPER))$

関係演算子 ⋈ (natural join)

- Bowtie ⋈
- 共通の属性名をもつタプルの全ての組み合わせ

JOURNAL

Journal	JournalName	Society
CACM	Communication of ACM	ACM
TODS	Transactions on Database Systems	ACM

ACM_PAPER

Author	Affiliation	Theory	Year	Journal
E. F. Codd	IBM	1NF	1970	CACM
M. Roth	UTA	¬1NF	1988	TODS
H. Korth	UTA	¬1NF	1988	TODS
A. Silberschatz	UTA	¬1NF	1988	TODS

関係演算子 ⋈ (natural join)

- ACM_PAPER ⋈ JOURNAL

Author	…	Year	Journal	JournalName	Society
E. F. Codd	…	1970	CACM	Communication of ACM	ACM
M. Roth	…	1988	TODS	Transactions on Database Systems	ACM
H. Korth	…	1988	TODS	Transactions on Database Systems	ACM
A. Silberschatz	…	1988	TODS	Transactions on Database Systems	ACM

関係演算子のまとめ

- Set union $R \cup S$
- Set intersection $R \cap S$
- Set difference $R - S$
- Projection $\pi_{attributes}(R)$
- Selection $\sigma_{conditions}(R)$
- Natural join $R \bowtie S$

- $\Rightarrow$ データ工学

データベースへの問い合わせ言語: SQL

SQL

- Structured Query Language
  - 英語圏では Sequel と発音される
- 起源：IBM System R
  - System R用のデータベース問い合わせ言語
- 今日：ISO/IEC 9075
  - ISO/IEC JTC1 SC 32 にて標準化・改訂作業

SQLの問い合わせと関係代数

SQL句	関係代数
UNION	$R \cup S$
INTERSECT	$R \cap S$
EXCEPT	$R - S$
SELECT	$\pi_{attributes}(R)$
WHERE	$\sigma_{conditions}(R)$

- 重複行の削除など、厳密には違いがある

SQL言語の概要

Query	• SELECT文 • 関係代数に相当
DML Data Manipulation Lang.	• 行の挿入、更新、削除 • トランザクションの記述
DDL Data Definition Language	• テーブルの生成 • テーブルの変更、消去
DCL Data Control Language	• アクセス権の付与 • アクセス権の取り消し
PSM Persistent Stored Module	• 手続き型プログラミング • 制御フロー

SQL文の例


```
CREATE TABLE products (
    product_no integer PRIMARY KEY,
    name text,
    price numeric
);
```

```
GRANT UPDATE ON accounts TO joe;
```

```
INSERT INTO products (product_no, name, price) VALUES
    (1, 'Cheese', 9.99),
    (2, 'Bread', 1.99),
    (3, 'Milk', 2.99);
```

```
SELECT product_id, p.name, (sum(s.units) * p.price) AS sales
    FROM products p LEFT JOIN sales s USING (product_id)
    GROUP BY product_id, p.name, p.price;
```

Source: PostgreSQL 9.0.4 Documentation

まとめ:データベースとSQL

- 並行制御
  - セマフォ等
  - トランザクション
- 関係代数: データベースの計算モデル
- データベースへの問い合わせ言語 SQL

演習 : PostgreSQLを使ってみよう

用意する環境

- 個人常用端末
- VMware Fusion
  - (個人常用端末に標準でインストールされている)
- VMイメージファイル

手順

- VMイメージのコピー
- VMの起動(OSの起動)
- ログイン
- DBのスーパーユーザになる
- DBの作成・接続
- テーブルの作成・操作
- いろいろなSQL文を試してみよう

1.準備

- VMイメージのコピー
  $ cp ~/../tomonori-i/debian.tar.gz　~/

- 展開
  $ tar xfz debian.tar.gz

- VMware Fusion を起動
- 「既存の仮想マシンを開きます。」を選択、
  展開したVMイメージ(debian)を選ぶ

#OSが立ち上がるとPostgreSQLも起動する

2.ログイン

- ログイン情報

  username: student
  password : system2011

- ログイン後，データベースのスーパーユーザpostgresになる

  $su postgres

  ※passwordは postgres

3.データベースの作成・接続

- データベースの作成

  $createdb dbname

- データベースへの接続

  $psql –d dbname

※dbnameは自由に決めてよい。
既にcomputersystemというDBが用意してあるのでこれを利用しても可。

4.テーブルの作成・操作

- 以下では、授業資料で示したテーブルを作成する。
- テーブルの作成

```
# CREATE TABLE acm_paper (Author text, Affiliation text, Theory
 text,Year  integer,Journal text);
```

- テーブルの操作
  - レコードの登録

```
# INSERT INTO acm_paper (Author , Affiliation ,
  Theory ,Year ,Journal)
    VALUES ('E.F.Codd','IBM','1NF', 1970 ,'CACM');
# …
```

  - 登録したレコードの確認

```
# SELECT * FROM acm_paper;
```

5. いろいろなSQL文を試してみよう

- UNION
- INTERSECT
- EXCEPT
- SELECT
- WHERE
- JOIN
- …

補足

- 参考ホームページ
  - http://homepage2.nifty.com/sak/w_sak3/doc/sysbrd/sak3sql.htm
- TAのメールアドレス
  - computer-system@is.naist.jp

補足資料

サンプルコード(1)

- ACM_Paper テーブルの作成

```
#CREATE TABLE acm_paper (Author text, Affiliation text,
   Theory text,Year Integer,Journal text);
#INSERT INTO acm_paper (Author , Affiliation , Theory ,Year ,Journal)
   VALUES ('E.F.Codd','IBM','1NF', 1970 ,'CACM');
#INSERT INTO acm_paper (Author , Affiliation , Theory ,Year ,Journal)
   VALUES ('M.Roth','UTA','¬1NF', 1988 ,'TODS');
#INSERT INTO acm_paper (Author , Affiliation , Theory ,Year ,Journal)
   VALUES ('H.Korth','UTA','¬1NF', 1988 ,'TODS');
#INSERT INTO acm_paper (Author , Affiliation , Theory ,Year ,Journal)
   VALUES ('A.Silberschatz','UTA','¬1NF', 1988 ,'TODS');
```

サンプルコード(2)

- IEEE_Paper テーブルの作成

```
#CREATE TABLE ieee_paper (Author text, Affiliation text, Theory text,
   Year integer, Journal text);
#INSERT INTO ieee_paper (Author , Affiliation , Theory ,Year ,Journal)
  VALUES ('S.Ceri','PDM','SQL', 1985 ,'TOSE');
#INSERT INTO ieee_paper (Author , Affiliation , Theory ,Year ,Journal)
   VALUES ('G.Gottlob','PDM','SQL', 1985 ,'TOSE');
```

サンプルコード(3)

- Journal テーブルの作成

```
# CREATE TABLE journal (Journal text, JournalName text, Society text);
# INSERT INTO journal (Journal , JournalName , Society) VALUES
  ('CACM','Communication of ACM','ACM');
# INSERT INTO journal (Journal , JournalName , Society) VALUES
  ('TODS','Transactions on Database Systems','ACM');
```

- Affiliation テーブルの作成

```
# CREATE TABLE affiliation (Affiliation text primary key, AffiliationName
   text, Country text);
# INSERT INTO affiliation (Affiliation, AffiliationName, Country ) VALUES
  ('IBM','IBM Research Laboratory','Etats-Unis');
# INSERT INTO affiliation (Affiliation, AffiliationName, Country ) VALUES
  ('UTA','University of Texas at Austin','Etats-Unis');
```

サンプルコード(4)

- 授業資料P.17　関係演算子 ∪ (set union)
- UNION
  - ACM_Paper ∪ IEEE_Paper

SELECT * FROM acm_paper UNION SELECT * from ieee_paper;
SELECT * FROM acm_paper UNION ALL SELECT * from ieee_paper;
※UNIONだと重複なし、UNION ALLだと重複あり

サンプルコード(5)

- 授業資料P.18 関係演算子 ∩ (set intersection)
- INTERSECT
  - ACM_Paper ∩ IEEE_Paper

 # SELECT * FROM acm_paper INTERSECT SELECT * from ieee_paper;
※例のテーブルには同じレコードが無いので追加して結果を見る
INSERT INTO ieee_paper VALUES ('A.Silberschatz','UTA','￢1NF', 1988 ,'TODS');

サンプルコード(6)

- 授業資料P.19 関係演算子 − (set difference)
- EXCEPT
  - ACM_Paper - IEEE_Paper

```
# SELECT * FROM acm_paper EXCEPT SELECT * from ieee_paper;
```

サンプルコード(7)

- 授業資料P.20 関係演算子 π (projection)
- SELECT
  - $\pi_{author}(IEEE\_PAPER)$

```
# SELECT author FROM ieee_paper;
```

  - $\pi_{author}(ACM\_PAPER) - \pi_{author}(IEEE\_PAPER)$

```
# SELECT author FROM acm_paper EXCEPT SELECT author
  FROM ieee_paper;
```

サンプルコード(8)

- 授業資料P.21 関係演算子 σ (selection)
- WHERE
  - $\sigma_{affiliation=‘IBM’}(IEEE\_PAPER)$

```
# SELECT * FROM ieee_paper WHERE affiliation = 'IBM';
```

  - $\pi_{author}(\sigma_{affiliation=‘IBM’ \wedge year < 1990}(IEEE\_PAPER \cup ACM\_PAPER))$

```
#  SELECT author FROM ( SELECT * FROM acm_paper EXCEPT
   SELECT * FROM  ieee_paper ) AS papars
   WHERE affiliation = 'IBM' and year < 1990 ;
```

サンプルコード(9)

- 授業資料 P.23 関係演算子 ⋈ (natural join)
- NATURAL JOIN
  - ACM_PAPER ⋈ JOURNAL

```
# SELECT * FROM acm_paper NATURAL JOIN journal;
```